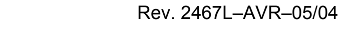
产品特点

- 高性能、低功耗的 AVR® 8 位微处理器
- 先进的 RISC 结构
  - 133 条指令 – 大多数可以在一个时钟周期内完成
  - 32 x 8 通用工作寄存器 + 外设控制寄存器
  - 全静态工作
  - 工作于 16 MHz 时性能高达 16 MIPS
  - 只需两个时钟周期的硬件乘法器
- 非易失性的程序和数据存储器
  - 128K 字节的系统内可编程 Flash
    寿命 : 10,000 次写 / 擦除周期
  - 具有独立锁定位、可选择的启动代码区
    通过片内的启动程序实现系统内编程
    真正的读 - 修改 - 写操作
  - 4K 字节的 EEPROM
    寿命 : 100,000 次写 / 擦除周期
  - 4K 字节的内部 SRAM
  - 多达 64K 字节的优化的外部存储器空间
  - 可以对锁定位进行编程以实现软件加密
  - 可以通过 SPI 实现系统内编程
- JTAG 接口 (与 IEEE 1149.1 标准兼容)
  - 遵循 JTAG 标准的边界扫描功能
  - 支持扩展的片内调试
  - 通过 JTAG 接口实现对 Flash, EEPROM, 熔丝位和锁定位的编程
- 外设特点
  - 两个具有独立的预分频器和比较器功能的 8 位定时器 / 计数器
  - 两个具有预分频器、比较功能和捕捉功能的 16 位定时器 / 计数器
  - 具有独立预分频器的实时时钟计数器
  - 两路 8 位 PWM
  - 6 路分辨率可编程 （2 到 16 位）的 PWM
  - 输出比较调制器
  - 8 路 10 位 ADC
    8 个单端通道
    7 个差分通道
    2 个具有可编程增益 （1x, 10x, 或 200x）的差分通道
  - 面向字节的两线接口
  - 两个可编程的串行 USART
  - 可工作于主机 / 从机模式的 SPI 串行接口
  - 具有独立片内振荡器的可编程看门狗定时器
  - 片内模拟比较器
- 特殊的处理器特点
  - 上电复位以及可编程的掉电检测
  - 片内经过标定的 RC 振荡器
  - 片内 / 片外中断源
  - 6 种睡眠模式 : 空闲模式、ADC 噪声抑制模式、省电模式、掉电模式、Standby 模式以及扩展的 Standby 模式
  - 可以通过软件进行选择的时钟频率
  - 通过熔丝位可以选择 ATmega103 兼容模式
  - 全局上拉禁止功能
- I/O 和封装
  - 53 个可编程 I/O 口线
  - 64 引脚 TQFP 与 64 引脚 MLF 封装
- 工作电压
  - 2.7 - 5.5V ATmega128L
  - 4.5 - 5.5V ATmega128
- 速度等级
  - 0 - 8 MHz ATmega128L
  - 0 - 16 MHz ATmega128



8 位 AVR® 微处理器，具有 128K 字节的系统内可编程 Flash

ATmega128 ATmega128L


Rev. 2467L–AVR–05/04

引脚配置

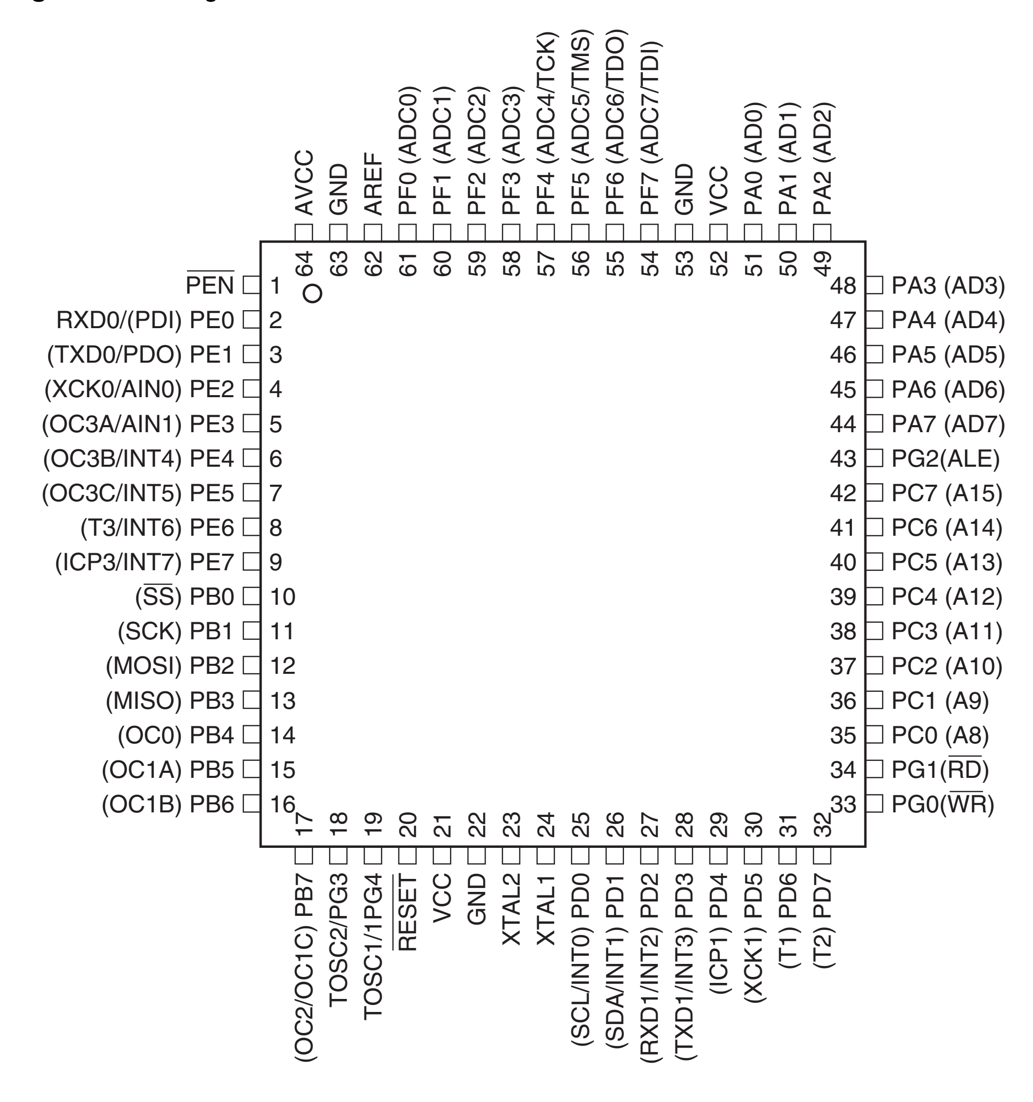


Figure 1. ATmega128 的引脚

综述

ATmega128为基于AVR RISC结构的8位低功耗CMOS微处理器。由于其先进的指令集以及单周期指令执行时间， ATmega128 的数据吞吐率高达 1 MIPS/MHz，从而可以缓减系统在功耗和处理速度之间的矛盾。

方框图

Figure 2. 方框图

PF0 - PF7 PA0 - PA7 PC0 - PC7 XTAL1 XTAL2 RESET
VCC GND
PORTF DRIVERS PORTA DRIVERS PORTC DRIVERS
DATA REGISTER PORTF DATA DIR. REG. PORTF DATA REGISTER PORTA DATA DIR. REG. PORTA DATA REGISTER PORTC DATA DIR. REG. PORTC
8-BIT DATA BUS
AVCC AGND AREF
ADC CALIB. OSC INTERNAL OSCILLATOR OSCILLATOR OSCILLATOR
JTAG TAP PROGRAM COUNTER STACK POINTER WATCHDOG TIMER
ON-CHIP DEBUG PROGRAM FLASH SRAM MCU CONTROL REGISTER TIMING AND CONTROL
BOUNDARY-SCAN INSTRUCTION REGISTER GENERAL PURPOSE REGISTERS X Y Z TIMER/ COUNTERS
PEN PROGRAMMING LOGIC INSTRUCTION DECODER INTERRUPT UNIT
CONTROL LINES ALU EEPROM
STATUS REGISTER
USART0 SPI USART1 TWO-WIRE SERIAL INTERFACE
ANALOG COMPARATOR
DATA REGISTER PORTE DATA DIR. REG. PORTE DATA REGISTER PORTB DATA DIR. REG. PORTB DATA REGISTER PORTD DATA DIR. REG. PORTD DATA REG. PORTG DATA DIR. REG. PORTG
PORTE DRIVERS PORTB DRIVERS PORTD DRIVERS PORTG DRIVERS
PE0 - PE7 PB0 - PB7 PD0 - PD7 PG0 - PG4

AVR 内核具有丰富的指令集和 32 个通用工作寄存器。所有的寄存器都直接与算逻单元 (ALU) 相连接，使得一条指令可以在一个时钟周期内同时访问两个独立的寄存器。这种结构大大提高了代码效率，并且具有比普通的复杂指令集微处理器高 10 倍的数据吞吐率。

ATmega128 具有如下特点:128K 字节的系统内可编程 Flash(具有在写的过程中还可以读的能力，即 RWW)、4K 字节的 EEPROM、4K 字节的 SRAM、53 个通用 I/O 口线、32 个通用工作寄存器、实时时钟 RTC、4 个灵活的具有比较模式和 PWM 功能的定时器 / 计数器 (T/C)、两个 USART、面向字节的两线接口 TWI、 8 通道 10 位 ADC(具有可选的可编程增益)、具有片内振荡器的可编程看门狗定时器、 SPI 串行端口、与 IEEE 1149.1 规范兼容的 JTAG 测试接口（此接口同时还可以用于片上调试），以及六种可以通过软件选择的省电模式。空闲模式时 CPU 停止工作，而 SRAM、 T/C、 SPI 端口以及中断系统继续工作；掉电模式时晶体振荡器停止振荡，所有功能除了中断和硬件复位之外都停止工作，寄存器的内容则一直保持；省电模式时异步定时器继续运行，以允许用户维持时间基准，器件的其他部分则处于睡眠状态； ADC 噪声抑制模式时 CPU 和所有的 I/O 模块停止运行，而异步定时器和 ADC 继续工作，以减少 ADC 转换时的开关噪声； Standby 模式时振荡器工作而其他部分睡眠，使得器件只消耗极少的电流，同时具有快速启动能力；扩展 Standby 模式则允许振荡器和异步定时器继续工作。

器件是以 Atmel 的高密度非易失性内存技术生产的。片内 ISP Flash 可以通过 SPI 接口、通用编程器，或引导程序多次编程。引导程序可以使用任何接口来下载应用程序到应用 Flash存储器。在更新应用Flash存储器时引导Flash区的程序继续运行，实现RWW操作。通过将 8 位 RISC CPU 与系统内可编程的 Flash 集成在一个芯片内， ATmega128 为许多嵌入式控制应用提供了灵活而低成本的方案。

ATmega128 AVR 有整套的开发工具，包括 C 编译器，宏汇编，程序调试器 / 仿真器和评估板。

ATmega103 与 ATmega128 的兼容性

ATmega128 是一个很复杂的微处理器，其 I/O 数目为 AVR 指令集所保留的 64 个 I/O 的超集。为了保持对 ATmega103 的兼容性， ATmega103 的 I/O 位置在 ATmega128 得到了保留。多数添加的 I/O 位于扩展的 I/O 空间 $60 到 $FF (即位于 ATmega103 的内部 RAM 空间)。这些地址可以通过指令 LD/LDS/LDD 和 ST/STS/STD 来访问，而不是 IN/OUT 指令。对于 ATmega103 用户而言，内部 RAM 可能还是个问题。此外，由于中断向量的增加，若程序使用了绝对地址可能也是个问题。为了解决这些问题， ATmega128 设置了一个熔丝位 M103C。此熔丝位编程后就可以使 ATmega128 工作于 ATmega103 兼容模式。此时扩展 I/O 空间将无法使用，而内部 RAM 正好与 ATmega103 的一致。同时扩展的中断向量也被取消了。

ATmega128 百分之百与 ATmega103 引脚兼容，可以在 PCB 上取代 ATmega103。应用手册 “Replacing ATmega103 by ATmega128” 详细告诉用户在用 ATmega128 取代 ATmega103 时需要注意的地方。

ATmega103 兼容模式

通过编程熔丝位 M103C，从 RAM、 I/O 引脚和中断向量的角度 ATmega128 将与 ATmega103 相兼容。但是， ATmega128 的一些新特点也就无法使用了。如下所示：

- 只剩下一个 USART，而且只支持异步模式。波特率寄存器只有低 8 位可用。
- 只有一个 16 位的定时器 / 计数器，两个比较寄存器，而不是两个 16 位定时器 / 计数器，三个比较寄存器。
- 不支持两线接口。
- 端口 C 只能输出。
- 端口 G 只能用做第二功能，而不能作为通用 I/O 端口。
- 端口 F 只能作为输入，而不能作为 ADC 的模拟输入引脚。
- 不支持引导程序功能。
- 不能够调节片内 RC 振荡器的频率。

- 外部存储器接口无法释放任何一个地址引脚作为通用 I/O，也不能够为不同的外部存储器地址区配置不同的等待周期。

下面的内容则使 ATmega128 更兼容 ATmega103：

- 在 MCUCSR 里只有 EXTRF 和 PORF。
- 改变看门狗溢出时间没有时序要求。
- 外部中断引脚 3 - 0 只能作为电平中断。
- USART 没有 FIFO 缓冲器。

在写操作中， ATmega103 没有使用的 I/O 应该写 0。

引脚说明

VCC

数字电路的电源。

GND

地。

端口 A(PA7..PA0)

端口 A 为 8 位双向 I/O 口，并具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，则端口被外部电路拉低时将输出电流。复位发生时端口 A 为三态。

端口 A 也可以用做其他不同的特殊功能，请参见 P 68。

端口 B(PB7..PB0)

端口 B 为 8 位双向 I/O 口，并具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，则端口被外部电路拉低时将输出电流。复位发生时端口 B 为三态。

端口 B 也可以用做其他不同的特殊功能，请参见 P 69。

端口 C(PC7..PC0)

端口 C 为 8 位双向 I/O 口，并具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，则端口被外部电路拉低时将输出电流。复位发生时端口 C 为三态。

端口 C 也可以用做其他不同的特殊功能，请参见 P 72。在 ATmega103 兼容模式下，端口 C 只能作为输出，而且在复位发生时不是三态。

端口 D(PD7..PD0)

端口 D 为 8 位双向 I/O 口，并具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，则端口被外部电路拉低时将输出电流。复位发生时端口 D 为三态。

端口 D 也可以用做其他不同的特殊功能，请参见 P 73。

端口 E(PE7..PE0)

端口 E 为 8 位双向 I/O 口，并具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，则端口被外部电路拉低时将输出电流。复位发生时端口 E 为三态。

端口 E 也可以用做其他不同的特殊功能，请参见 P 75。

端口 F(PF7..PF0)

端口 F 为 ADC 的模拟输入引脚。

如果不作为 ADC 的模拟输入，端口 F 可以作为 8 位双向 I/O 口，并具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，则端口被外部电路拉低时将输出电流。复位发生时端口 F 为三态。如果使能了 JTAG 接口，则复位发生时引脚 PF7(TDI)、 PF5(TMS) 和 PF4(TCK) 的上拉电阻使能。

端口 F 也可以作为 JTAG 接口。



在 ATmega103 兼容模式下，端口 F 只能作为输入引脚。

端口 G(PG4..PG0)

端口 G 为 5 位双向 I/O 口，并具有可编程的内部上拉电阻。其输出缓冲器具有对称的驱动特性，可以输出和吸收大电流。作为输入使用时，若内部上拉电阻使能，则端口被外部电路拉低时将输出电流。复位发生时端口 G 为三态。

端口 G 也可以用做其他不同的特殊功能。

在 ATmega103 兼容模式下，端口 G 只能作为外部存储器的所存信号以及 32 kHz 振荡器的输入，并且在复位时这些引脚初始化为 PG0 = 1， PG1 = 1 以及 PG2 = 0。 PG3 和 PG4 是振荡器引脚。

$\overline{\text{RESET}}$

复位输入引脚。超过最小门限时间的低电平将引起系统复位。门限时间在 P 47Table 19 说明。低于此时间的脉冲不能保证可靠复位。

XTAL1

反向振荡器放大器及片内时钟操作电路的输入。

XTAL2

反向振荡器放大器的输出。

AVCC

AVCC 为端口 F 以及 ADC 转换器的电源，需要与 V_{CC} 相连接，即使没有使用 ADC 也应该如此。使用 ADC 时应该通过一个低通滤波器与 V_{CC} 连接。

AREF

AREF 为 ADC 的模拟基准输入引脚。

PEN

PEN 是 SPI 串行下载的使能引脚。在上电复位时保持 $\overline{\text{PEN}}$ 为低电平将使器件进入 SPI 串行下载模式。在正常工作过程中 $\overline{\text{PEN}}$ 引脚没有其他功能。

代码例子

本手册包含了一些简单的代码例子以说明如何使用芯片各个不同的功能。这些例子都假定在编译之前已经包含了正确的头文件。有些 C 编译器在头文件里并没有包含位定义和中断，而且各个 C 编译器对中断处理有自己不同的处理方式。请注意查阅其文档以获取具体的信息。

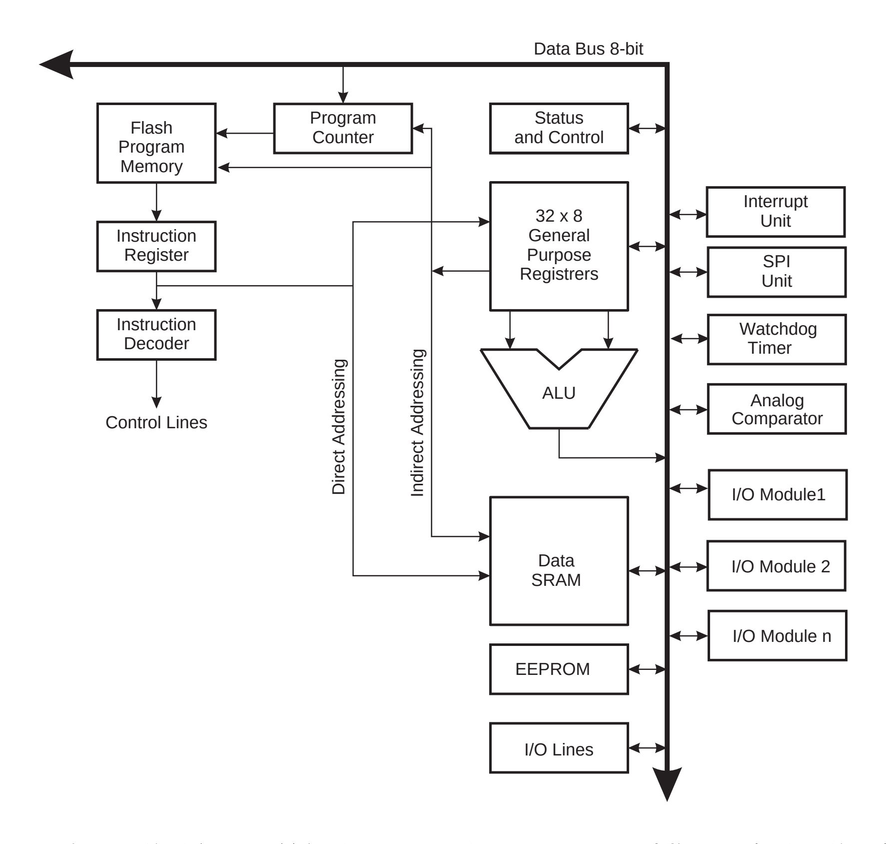
AVR CPU 内核

介绍

本节从总体上讨论 AVR 内核的结构。CPU 的主要任务是保证程序的正确执行。因此它必须能够访问存储器，执行运算，控制外设以及处理中断。

结构综述

Figure 3. AVR 结构的方框图



为了得到最大程度的性能以及并行性， AVR 采用了 Harvard 结构，具有独立的数据和程序总线。程序存储器的指令通过一级流水线运行。 CPU 在执行一条指令的同时读取下一条指令 (在本文称为预取)。这个概念实现了指令的单时钟周期运行。程序存储器为可以在线编程的 FLASH。

快速访问寄存器文件包括32个8位通用工作寄存器，而且都可以在一个时钟周期内访问。从而实现单时钟周期的 ALU 操作。在典型的 ALU 操作过程中，两个位于寄存器文件中的操作数同时被访问，然后执行相应的运算，结果再被送回寄存器文件。整个过程仅需要一个时钟周期。

寄存器文件里有 6 个寄存器可以用作 3 个 16 位的间接地址寄存器指针以寻址数据空间，实现高效的地址运算。其中一个指针还可以作为程序存储器查询表的地址指针。这些附加的功能寄存器即为 16 位的 X、 Y、 Z 寄存器。

ALU支持寄存器之间以及寄存器和常数之间的算术和逻辑运算。ALU也可以执行单寄存器操作。运算完成之后状态寄存器的内容将更新以反映操作结果。

程序流程通过有 / 无条件的跳转指令和调用指令来控制，从而直接寻址整个地址空间。大多数指令长度为 16 位，亦即每个程序存储器地址都包含一条 16 位或 32 位的指令。

程序存储器空间分为两个区：引导程序区和应用程序区。这两个区都有专门的锁定位以实现读和读 / 写保护。用于写应用程序区的 SPM 指令必须位于引导程序区。



在中断和调用子程序时返回地址程序计数器 (PC) 保存于堆栈之中。堆栈位于通用数据 SRAM，故此其深度仅受限于 SRAM 的大小。在复位例程里用户首先要初始化堆栈指针 SP。这个指针位于 I/O 空间，可以进行读写访问。数据 SRAM 可以通过 5 种不同的寻址模式进行访问。

AVR 存储器空间为线性的平面结构。

AVR 具有一个灵活的中断模块。控制寄存器位于 I/O 空间。状态寄存器里有全局中断使能位。在程序存储器起始处有一个中断向量表，每一个中断在此都有独立的中断向量。各个中断的优先级与其在中断向量表的位置有关，中断向量地址越低，优先级越高。

I/O 存储器空间包含 64 个可以直接寻址的地址。映射到数据空间即为寄存器文件之后的地址 $20 - $5F。此外，ATmega128 在 SRAM 里还有扩展的 I/O 空间，位于地址 $60 - $FF。但是只能使用 ST/STS/STD 和 LD/LDS/LDD 指令。

ALU - 算逻单元

AVR ALU 与 32 个通用工作寄存器直接相连。寄存器与寄存器之间、寄存器与立即数之间的 ALU 运算只需要一个时钟周期。 ALU 操作分为 3 类：算术、逻辑和位操作。此外还提供了支持无 / 有符号数和分数乘法的乘法器。具体请参见指令集。

状态寄存器

状态寄存器包含了最近执行的算术指令的结果信息。这些信息可以用来改变程序流程以实现条件操作。状态寄存器的内容只有在 ALU 运算结束后才会更新。这样，在多数情况下就不需要专门的比较指令了，从而使系统运行更快速，代码效率更高。

在进入中断例程时状态寄存器不会自动保存；中断返回时也不会自动恢复。这些工作需要软件来处理。

AVR 中断寄存器 – SREG – 定义如下：

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – I: 全局中断使能**

置位时使能全局中断。单独的中断使能由其他独立的控制寄存器控制。如果 I 清零，则不论单独中断标志置位与否，都不会产生中断。任意一个中断发生后 I 清零，而执行 RETI 指令后置位以使能中断。 I 也可以通过 SEI 和 CLI 指令来置位和清零。

- **Bit 6 – T: 位拷贝存储**

位拷贝指令 BLD 和 BST 利用 T 作为目的或源地址。BST 把寄存器的某一位拷贝到 T，而 BLD 把 T 拷贝到寄存器的某一位。

- **Bit 5 – H: 半进位标志**

半进位标志 H 表示算术操作发生了半进位。此标志对于 BCD 运算非常有用。

- **Bit 4 – S: 符号位，$S = N \oplus V$**

S 为负数标志 N 与 2 的补码溢出标志 V 的异或。

- **Bit 3 – V: 2 的补码溢出标志**

支持 2 的补码运算。

- **Bit 2 – N: 负数标志**

表明算术或逻辑操作结果为负。

- **Bit 1 – Z: 零标志**

表明算术或逻辑操作结果为零。

- **Bit 0 – C: 进位标志**

表明算术或逻辑操作发生了进位。

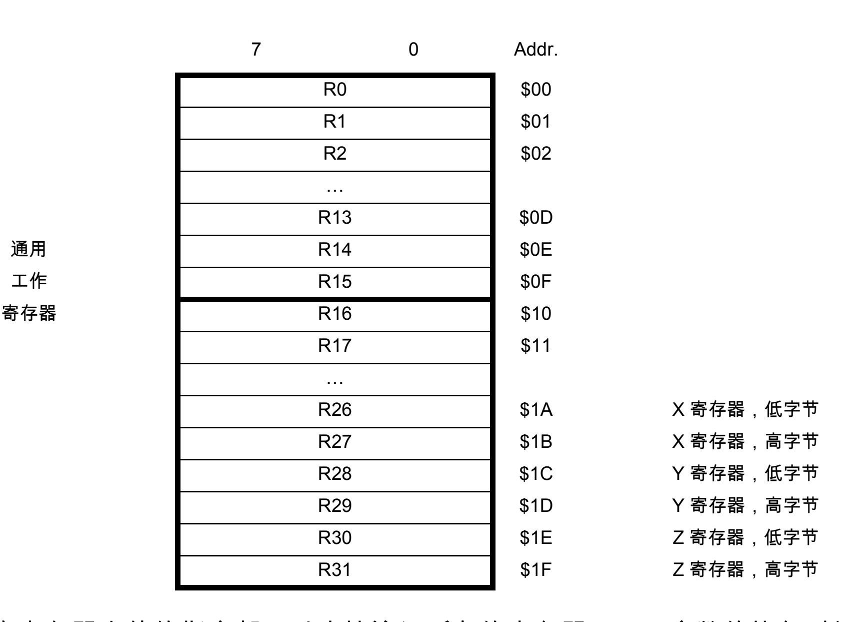
通用寄存器文件

寄存器文件针对 AVR 增强型 RISC 指令集做了优化。为了获得需要的性能和灵活性，寄存器文件支持以下的输入 / 输出方案：

- 一个 8 位输出操作数和一个 8 位结果输入。
- 两个 8 位位输出操作数和一个 8 位结果输入。
- 两个 8 位位输出操作数和一个 16 位结果输入。
- 一个 16 位位输出操作数和一个 16 位结果输入。

Figure 4 为 CPU 32 个通用工作寄存器的结构。

Figure 4. AVR CPU 通用工作寄存器



大多数操作寄存器文件的指令都可以直接访问所有的寄存器，而且多数的执行时间为单时钟周期。

如 Figure 4 所示，每个寄存器都有一个数据内存地址，将他们直接映射到用户数据空间的头 32 个地址。虽然寄存器文件的物理实现不是 SRAM，这种内存组织方式在访问寄存器方面具有极大的灵活性，因为 X、 Y、 Z 寄存器可以设置为指向任意寄存器的指针。

X 寄存器 ,Y 寄存器和 Z 寄存器

寄存器 R26..R31 除了用作通用寄存器外，还可以作为数据间接寻址用的地址指针。这三个间接寻址寄存器示于 Figure 5。

Figure 5. X、 Y、 Z 寄存器





Z 寄存器

7	0	7	0
R31 ($1F)		R30 ($1E)	

在不同的寻址模式中，这些地址寄存器可以实现固定偏移量，自动加一和自动减一功能。

堆栈指针

堆栈指针主要用来保存临时数据，局部变量和中断 / 自程序的返回地址。堆栈指针总是指向堆栈的顶部。要注意 AVR 的堆栈是向下生长的，即新数据推入堆栈时，堆栈指针的数值将减小。

堆栈指针指向位于 SRAM 的函数及中断堆栈。堆栈空间必须在调用函数或中断使能之前定义。指针必须指向高于 $60 的地址。用 PUSH 指令推数据入栈时，堆栈指针将减一，而当调用函数或中断时，指针将减二。使用 POP 指令时，堆栈指针将加一，而用 RET 或 RETI 返回时，指针将加二。

AVR 堆栈指针占用了 I/O 空间两个 8 位寄存器。使用的位数由实际情况决定。注意，在 AVR 结构中某些操作使用的数据空间很小，只要 SPL 即可，此时，不会给出 SPH 寄存器。

Bit	15	14	13	12	11	10	9	8	
	SP15	**SP14**	**SP13**	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
	SP7	**SP6**	**SP5**	**SP4**	**SP3**	**SP2**	**SP1**	**SP0**	**SPL**
	7	6	5	4	3	2	1	0	
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

RAM 页面的 Z 选择寄存器 - RAMPZ

Bit	7	6	5	4	3	2	1	0	
	–	—	–	–	–	–	–	**RAMPZ0**	**RAMPZ**
读 / 写	R	R	R	R	R	R	R	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bits 7..2 – Res: 保留**

保留位，读操作返回值为零。在写数据时要写入 0 以保证与未来产品的兼容。

- **Bit 1 – RAMPZ0: 扩展 RAM 页面 Z 指针**

RAMPZ 寄存器用于选择 Z 指针访问的是哪一个 64K RAM。由于 ATmega128 不支持超过 64K 的存储器，因此 RAMPZ 只用来协助 ELPM/SPM 指令决定访问哪一个程序存储器页。不同的 RAMPZ0 的作用如下：

RAMPZ0 = 0: ELPM/SPM 可以访问程序存储器地址 $0000 - $7FFF（低 64K 字节）

RAMPZ0 = 1: ELPM/SPM 可以访问程序存储器地址 $8000 - $FFFF（高 64K 字节）

LPM 不受 RAMPZ 设置的影响。

指令执行时序

这一节介绍指令执行和内存访问时序。AVR CPU 由系统时钟 clk_{CPU} 驱动。此时钟由外部晶体直接产生。芯片内没有时钟分频。

Figure 6 说明了由 Harvard 结构决定的并行取指和指令执行，以及快速访问寄存器文件的概念。这是一个基本的、达到 1 MIPS/MHz，具有优良的性价比、功能 / 时钟比、功能 / 功耗比的流水线概念。





Figure 6. 并行取指和执行

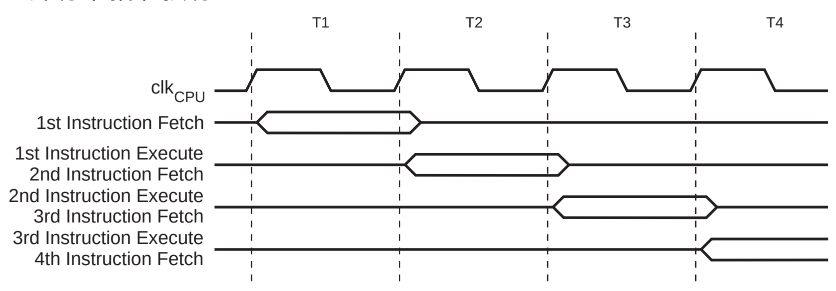
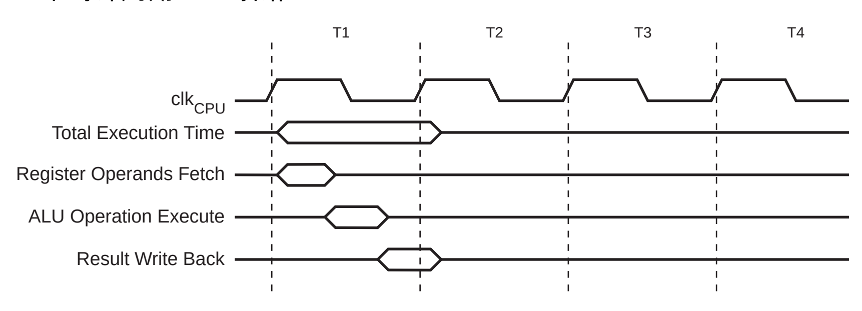


Figure 7 演示的是寄存器文件内部时序。在一个时钟周期里，ALU 可以同时对两个寄存器操作数进行操作，同时将结果存回到其中的一个寄存器中去。

Figure 7. 单时钟周期 ALU 操作



复位和中断处理

AVR 有不同的中断源。每个中断和复位在程序空间都有一个独立的中断向量。所有的中断事件都有自己的使能位。当使能位置位，且状态寄存器的全局中断使能位 I 也置位的情况下，中断可以发生。根据不同的程序计数器 PC 数值，在引导锁定位 BLB02 或 BLB12 被编程的情况下，中断可能自动禁止。这个特性提高了软件的安全性。具体请参见 P 267“存储器编程”。

程序存储器空间的最低地址缺省定义为复位和中断向量。完全的向量列表请参见 P 55“中断”。列表也决定了不同中断的优先级。向量所在的地址越低，优先级越高。 RESET 具有最高的优先级，下一个则为 INT0 – 外部中断请求 0。通过置位 MCU 控制寄存器 (MCUCR) 的 IVSEL，中断向量可以移至引导 Flash 的起始处 P 55“中断”一节有详细说明。编程熔丝位 BOOTRST 可以将复位向量也移至引导 Flash 的起始处。具体可参见 P 255“支持引导装入程序 – 在写的同时可以读 (RWW, Read-While-Write) 的自我编程能力”。

当中断发生时全局中断使能位 I 被清零，所有的中断都被禁止。用户软件可以通过置位位 I 来使能中断嵌套。此时所有的中断都可以中断当前中断。执行 RETI 指令后 I 自动置位。

从根本上说有两种类型的中断。第一种由事件触发并置位中断标志。对于这些中断，程序计数器跳转到实际的中断向量以执行中断处理例程，同时硬件将清除相应的中断标志。中断标志也可以通过对其写 ”1” 来清除。当中断发生后，如果相应的中断使能位为 ”0”，则中断标志位置位，并一直保持到中断执行，或者被软件清除。类似的，如果全局中断标志被清零，则所有以发生的中断都不会被执行，直到 I 置位。然后被挂起的各个中断按中断优先级依次执行。

第二种类型的中断则是只要中断条件满足，就会一直触发。这些中断不需要中断标志。若中断条件在中断使能之前就消失了，则中断不会被触发。

AVR 退出中断后总是回到主程序并执行一条指令才可以去执行其他被挂起的中断。

进入中断例程时状态寄存器不会自动保存；中断返回时也不会自动恢复。这些工作必须由软件来完成。

使用 CLI 指令来禁止中断时，中断立即禁止。没有中断可以在执行 CLI 指令后发生，即使它是在执行 CLI 的同时发生的。下面的例子说明了如何在写 EEPROM 时使用这个指令来防止中断发生。

汇编代码例程

```
in  r16, SREG      ; 保存SREG
cli     ; 禁止中断
sbi EECR, EEMWE    ; 启动 EEPROM 写操作
sbi EECR, EEWE
out SREG, r16      ; 恢复 SREG (I-bit)
```

C 代码例程

```
char cSREG;
cSREG = SREG;  /* 保存 SREG */
/* 禁止中断 */
_CLI();
EECR |= (1<<EEMWE); /* 启动 EEPROM 写操作 */
EECR |= (1<<EEWE);
SREG = cSREG; /* 恢复 SREG (I-bit) */
```

使用 SEI 指令使能中断时，紧跟其后的第一条指令在执行任何中断之前先执行。

汇编代码例程

```
sei   ; 置位全局中断使能标志
sleep ; 进入睡眠模式，等待中断发生
; 注意： 在执行任何被挂起的中断之前首先进入睡眠模式
```

C 代码例程

```
_SEI(); /* 置位全局中断使能标志 */
_SLEEP(); /* 进入睡眠模式，等待中断发生 */
/* 注意： 在执行任何被挂起的中断之前首先进入睡眠模式 */
```

中断响应时间

AVR 中断响应时间最少为 4 个时钟周期。4 个时钟周期后，程序跳转到实际的中断处理例程。在这 4 个时钟期间，PC 自动入栈。在通常情况下，中断向量为一个跳转指令，此跳转要花 3 个时钟周期。如果中断在一个多周期指令执行期间发生，则在此多周期指令执行完后 MCU 才会执行中断程序。若中断发生时 MCU 处于睡眠模式，中断响应时间增加到 8 个时钟周期。增加的时钟周期是由于唤醒启动时间引入的。

中断返回亦需 4 个时钟。在此期间 PC(两个字节) 将被弹出栈，堆栈指针加二，状态寄存器 SREG 的 I 置位。

AVR ATmega128 存储器

本节讲述 ATmega128 的存储器。 AVR 结构具有两个主存储器空间：数据寄存器和程序寄存器。此外， ATmega128 还有 EEPROM 存储器以保存数据。这三个存储器空间都是线性的。

系统内可编程的 Flash 程序存储器

ATmega128具有128K字节的在线编程Flash。因为所有的AVR指令为16位或32位，故尔 FLASH 组织成 64K x 16 的形式。考虑到软件安全性， Flash 程序存储器分为两个区：引导程序区和应用程序区。

Flash存储器至少可以擦写10,000 次。ATmega128的程序计数器PC为16位，因此可以寻址 64K 的程序存储器。引导程序区以及软件安全引导锁定位在 P 255“ 支持引导装入程序 – 在写的同时可以读 (RWW, Read-While-Write) 的自我编程能力 ” 有详细说明。而 P 267“ 存储器编程 ” 则介绍了利用 SPI 或 JTAG 接口实现对 Flash 的串行下载。

常数可以保存于整个程序存储器地址空间 (参见 LPM – 加载程序存储器指令以及 ELPM – 扩展的加载程序存储器指令)。

取指和指令执行时序图请参见 P 11“ 指令执行时序 ” 。

Figure 8. 程序存储器映像

SRAM 数据存储器

ATmega128 支持两种不同的 SRAM 配置，如 Table 1 所示。

Table 1. 存储器配置

配置	内部 SRAM 数据存储器	扩展的 SRAM 数据存储器
普通模式	4096	达到 64K
ATmega103 兼容模式	4000	达到 64K

Figure 9 说明了 ATmega128 的 SRAM 存储器是如何组织的。

ATmega128是一个复杂的微处理器，其支持的外设要比预留的64个I/O(通过IN/OUT指令访问) 所能支持的要多。对于扩展的 I/O 空间 $60 - $FF，只能使用 ST/STS/STD 和



LD/LDS/LDD 指令。当 ATmega128 工作于 ATmega103 兼容模式时，扩展的 I/O 将无法访问。

在普通模式下，前 4352 个数据地址包含寄存器文件， I/O 存储器，扩展的 I/O 存储器以及内部数据 SRAM。起始的 32 个地址为寄存器文件，然后是 64 个 I/O 存储器，接着是 160 个扩展的 I/O 存储器，最后是 4096 字节的内部数据 SRAM。

在 ATmega103 兼容模式下，前 4096 个数据地址包含寄存器文件， I/O 存储器以及内部数据 SRAM。起始的 32 个地址为寄存器文件，然后是 64 个 I/O 存储器，最后是 4000 字节的内部数据 SRAM。

ATmega128 还可以访问直到 64K 的外部数据 SRAM。其起始紧跟在内部 SRAM 之后。在普通模式下，寄存器文件、I/O 存储器、扩展的 I/O 存储器以及内部数据 SRAM 占据了低 4352 字节；而在 ATmega103 兼容模式下占据了 4096 字节 (没有扩展 I/O)。因此，在使用外部存储器时普通模式只能有 61184 字节，ATmega103 兼容模式只能有 61440 字节。具体请参见 P 23" 外部存储器接口 " 。

当访问 SRAM 的地址超出内部 SRAM 的地址时， MCU 将对外部 SRAM 寻址 （ 指令相同)。访问内部 SRAM 时读 / 写锁存信号 (PG0 和 PG1) 无效。若要访问外部 SRAM，必须置位 MCUCR 的 SRE。

访问外部 SRAM 比访问内部的要多一个时钟周期，这意味着 LD、 ST、 LDS、 STS、LDD、STD、PUSH 和 POP 指令将多一个时钟周期。如果堆栈放置于外部 SRAM，则中断和函数调用将花费额外的三个时钟周期。如果外部 SRAM 接口使用了 1、2、3 个等待周期，则访问周期将相应增加 2、3、4 个时钟周期；中断和子程序调用的开销则增加 5、7、 9 个时钟周期。

数据寻址模式分为 5 种：直接寻址，带偏移量的间接寻址，间接寻址，预减的间接寻址，以及后加的间接寻址。寄存器 R26 到 R31 为间接寻址的指针寄存器。

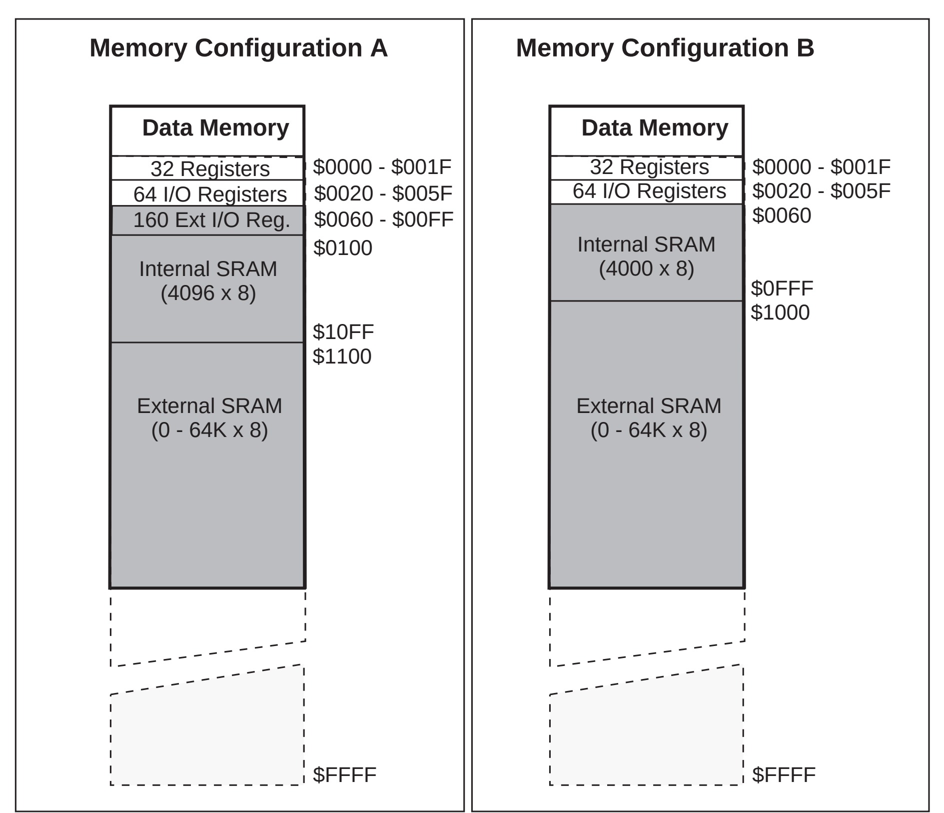
直接寻址访问整个数据空间。

带偏移量的间接寻址模式寻址到 Y、 Z 指针给定地址附近的 63 个地址。

带预减和后加的间接寻址模式要用到 X、 Y、 Z 指针。

32 个通用寄存器，64 个 I/O 寄存器，4096 字节的 SRAM 可以被所有的寻址模式所访问。寄存器文件说明见 P 9" 通用寄存器文件 " 。

Figure 9. 数据存储器映像

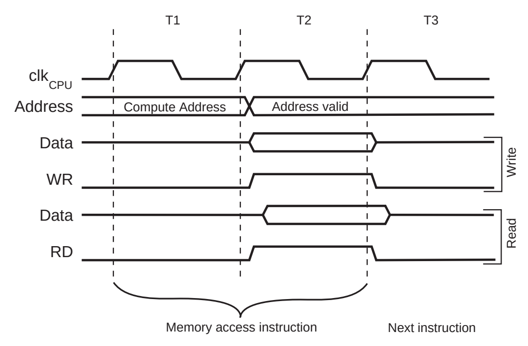




数据存储器访问时间

本节说明访问内部存储器的时序。如 Figure 10 所示，内部数据 SRAM 访问时间为两个 clk_{CPU} 时钟。

Figure 10. 内部数据 SRAM 访问周期



EEPROM 数据存储器

ATmega128包含4K字节的EEPROM。它是作为一个独立的数据空间而存在的，可以按字节读写。EEPROM 的寿命至少为 100,000 次（擦除）。EEPROM 的访问由地址寄存器，数据寄存器和控制寄存器决定

具体的 SPI 和 JTAG 下载 EEPROM 数据请分别参见 P 267“ 存储器编程 ” 。

EEPROM 读 / 写访问

EEPROM 的访问寄存器位于 I/O 空间。

EEPROM 的写访问时间由 Table 2 给出。自定时功能可以让用户监测何时开始写下一字节。如果用户要操作 EEPROM，应当注意如下问题：在电源滤波时间常数比较大的电路中，上电 / 下电时 V_{CC} 上升 / 下降速度会比较慢。此时 CPU 将工作于低于晶振所要求的电源电压。请参照 P 22 “ 防止 EEPROM 数据丢失 ” 以防止如何在此时出现 EEPROM 的数据丢失问题。

为了防止无意识的 EEPROM 写操作，需要执行一个特定的写时序。具体参看 EEPROM 控制寄存器的内容。

当执行 EEPROM 读操作时， CPU 会停止工作 4 个周期，然后再执行后续指令；当执行 EEPROM 写操作时， CPU 会停止工作 2 个周期，然后再执行后续指令。

EEPROM 地址寄存器 - EEARH 和 EEARL

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	EEAR11	EEAR10	EEAR9	EEAR8	EEARH
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
读 / 写	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	X	X	X	X	
	X	X	X	X	X	X	X	X	

- **Bits 15..12 – Res: 保留**

保留位，读操作返回值为零。在写数据时要写入 0 以保证与未来产品的兼容。

- **Bits 11..0 – EEAR11..0: EEPROM 地址**

EEARH 和 EEARL 指定了 4K 字节的 EEPROM 空间。EEPROM 的地址是线性的，从 0 到 4096。 EEAR 的初始值没有定义。在访问 EEPROM 之前必须为其赋予正确的数据。

EEPROM 数据寄存器 - EEDR

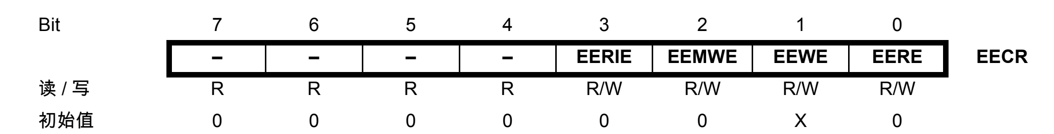
Bit	7	6	5	4	3	2	1	0

	MSB							LSB	EEDR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

• **Bits 7..0 – EEDR7.0: EEPROM 数据**

对于 EEPROM 写操作， EEDR 是需要写到 EEAR 单元的数据；对于读操作， EEDR 是从地址 EEAR 读取的数据。

EEPROM 控制寄存器 - EECR



Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
读 / 写	R	R	R	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	X	0	

• **Bits 7..4 – Res: 保留**

保留位，读操作返回值为零。

• **Bit 3 – EERIE: EEPROM 就绪中断使能**

若 SREG 的 I 为 "1“，则置位 EERIE 使能 EEPROM 就绪中断。清零 EERIE 则禁止此中断。当 EEWE 清零时 EEPROM 就绪中断即可发生。

• **Bit 2 – EEMWE: EEPROM 主机写使能**

EEMWE决定设置EEWE为"1“是否可以启动EEPROM写操作。。当EEMWE为"1“时，在4 个时钟周期内置位 EEWE 将把数据写入 EEPROM 的指定地址；若 EEMWE 为 "0“，则 EEWE 不起作用。 EEMWE 置位后 4 个周期，硬件对其清零。

• **Bit 1 – EEWE: EEPROM 写使能**

当 EEPROM 数据和地址设置好之后，需置位 EEWE 以便将数据写入 EEPROM。此时 EEMWE必须置位，否则EEPROM写操作将不会发生。写时序如下(第3和第4步不是必须的)：

1. 等待 EEWE 为 0。
2. 等待 SPMCSR 寄存器的 SPMEN 为零。
3. 将新的 EEPROM 地址写入 EEAR。
4. 将新的 EEPROM 数据写入 EEDR。
5. 对 EECR 寄存器的 EEMWE 写 "1“，同时清零 EEWE。
6. 在置位 EEMWE 的 4 个周期内，置位 EEWE。

在 CPU 写 Flash 存储器的时候不能对 EEPROM 进行编程。在启动 EEPROM 写操作之前软件必须要检查 Flash 写操作是否已经完成。第二步仅在软件包含引导程序，允许 CPU 对 Flash 进行编程时才有用。如果 CPU 永远都不会写 Flash，则第二步可以忽略。请参考 P 255“ 支持引导装入程序 – 在写的同时可以读 (RWW, Read-While-Write) 的自我编程能力 ” 。

注意：如有中断发生于步骤 5 和 6 之间将导致写操作失败。因为此时 EEPROM 写使能操作将超时。如果一个操作 EEPROM 的中断打断了另一个 EEPROM 操作，EEAR 或 EEDR 寄存器可能被修改，引起 EEPROM 操作失败。建议此时关闭全局中断标志 I。

经过写访问时间之后， EEWE 硬件清零。用户可以凭此位判断写时序是否已经完成。EEWE 置位后， CPU 要停止两个时钟周期才会运行下一条指令。

• **Bit 0 – EERE: EEPROM 读使能**

当 EEPROM 地址设置好之后，需置位 EERE 以便将数据读入 EEAR。EEPROM 数据的读取只需要一条指令，且无需等待。读取 EEPROM 时 CPU 要停止 4 个时钟周期。

用户在读取 EEPROM 时应该检测 EEWE。如果一个写操作正在进行，就无法读取 EEPROM，也无法改变寄存器 EEAR。

标定振荡器用于 EEPROM 访问定时。 Table 2 为 CPU 访问 EEPROM 的典型时间。

Table 2. EEPROM 编程时间

符号	标定的 RC 振荡器周期数 (1)	典型编程时间
EEPROM 写操作 (自 CPU)	8448	8.5 ms

Note: 1. 使用的是 1 MHz 的时钟。与熔丝位 CKSEL 的设置无关。

下面的代码分别用汇编和 C 函数说明如何实现 EEPROM 的写操作。在此假设中断不会在执行这些函数的过程当中发生。例子同时还假设软件没有引导程序。若引导程序存在，则 EEPROM 写函数还需要等待正在进行的 SPM 命令的结束。

汇编代码例程

```
EEPROM_write:
  ; 等待上一次写操作结束
  sbic EECR,EEWE
  rjmp EEPROM_write
  ; 设置地址寄存器 (r18:r17)
  out  EEARH, r18
  out  EEARL, r17
  ; 写数据到数据寄存器 (r16)
  out  EEDR,r16
  ; 置位 EEMWE
  sbi  EECR,EEMWE
  ; 置位 EEWE 以启动写操作
  sbi  EECR,EEWE
  ret
```

C 代码例程

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
  /* 等待上一次写操作结束 */
  while(EECR & (1<<EEWE))
    ;
  /* 设置地址和数据寄存器 */
  EEAR = uiAddress;
  EEDR = ucData;
  /* 置位 EEMWE */
  EECR |= (1<<EEMWE);
  /* 置位 EEWE 以启动写操作 E */
  EECR |= (1<<EEWE);
}
```



下一个代码例子说明如何用汇编和 C 来读取 EEPROM 在此假设中断不会在执行这些函数的过程当中发生。

汇编代码例程

```
EEPROM_read:
  ; 等待上一次写操作结束
  sbic EECR,EEWE
  rjmp EEPROM_read
  ; 设置地址寄存器 (r18:r17)
  out  EEARH, r18
  out  EEARL, r17
  ; 设置 EERE 以启动读操作
  sbi  EECR,EERE
  ; 自数据寄存器读取数据
  in   r16,EEDR
  ret
```

C 代码例程

```
unsigned char EEPROM_read(unsigned int uiAddress)
{
  /* 等待上一次写操作结束 */
  while(EECR & (1<<EEWE))
    ;
  /* 设置地址寄存器 */
  EEAR = uiAddress;
  /* 设置 EERE 以启动读操作 */
  EECR |= (1<<EERE);
  /* 自数据寄存器返回数据 */
  return EEDR;
}
```

掉电休眠模式下 EEPROM 的写入

当 EEPROM 执行写操作时进入掉电休眠模式， EEPROM 写操作继续，并在写访问时间结束前完成。但写操作结束后，振荡器继续工作，因此器件无法完全进入掉电模式。因此建议在进入掉电模式前检验 EEPROM 写操作是否完成。

防止 EEPROM 数据丢失

由于电源电压过低， CPU 和 EEPROM 有可能工作不正常，造成 EEPROM 数据的毁坏 (丢失)。这种情况在使用独立的 EEPROM 器件时也会遇到。

由于电压过低造成 EEPROM 数据损坏有两种可能：一是电压低于 EEPROM 写操作所需要的最低电压；二是 CPU 本身已经无法正常工作。

EEPROM 数据损坏的问题可以通过以下方法解决：

当电压过低时保持 AVR RESET 信号为低。这可以通过使能芯片的掉电检测电路 BOD 来实现。如果 BOD 电平无法满足要求，则可以使用外部复位电路。若写操作过程当中发生了复位，写操作将终止。

I/O 存储器

ATmega128 的 I/O 空间定义见 P 342“ 寄存器概述 ”。

ATmega128 的所有 I/O 和外设都被放置在 I/O 空间。所有的 I/O 地址都可以通过 LD/LDS/LDD 和 ST/STS/STD 指令来访问，在 32 个通用工作寄存器和 I/O 之间传输数据。地址为 $00 - $1F 的 I/O 寄存器还可用 SBI 和 CBI 指令直接进行位寻址，而 SBIS 和 SBIC 则用来检查单个位置位与否。当使用 IN 和 OUT 指令时地址必须在 $00 - $3F 之间。如果要象 SRAM 一样通过 LD 和 ST 指令访问 I/O 寄存器，相应的地址要加上 $20。ATmega128

是一个复杂的微处理器，其支持的外设要比预留的 64 个 I/O(通过 IN/OUT 指令访问) 所能支持的要多。对于扩展的 I/O 空间 $60 - $FF，只能使用 ST/STS/STD 和 LD/LDS/LDD 指令。当 ATmega128 工作于 ATmega103 兼容模式时，扩展的 I/O 被 SRAM 所取代。

为了与后续产品兼容，保留未用的未应写 "0“，而保留的 I/O 寄存器则不应进行写操作。

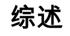
一些状态标志位的清除是通过写 "1“ 来实现的。CBI 和 SBI 指令可以操作 I/O 寄存器所有的位，并给置位的位回写 "1“，因此会清除这些标志位。 CBI 和 SBI 指令只对 $00 to $1F 之间的寄存器有效。

I/O 和外设控制寄存器在其他章节介绍。

外部存储器接口

由于外部存储器接口所提供的特性，此接口非常适合于与存储器器件互连，如外部 SRAM 和 Flash， LCD， A/D， D/A，等等。其主要特点为：

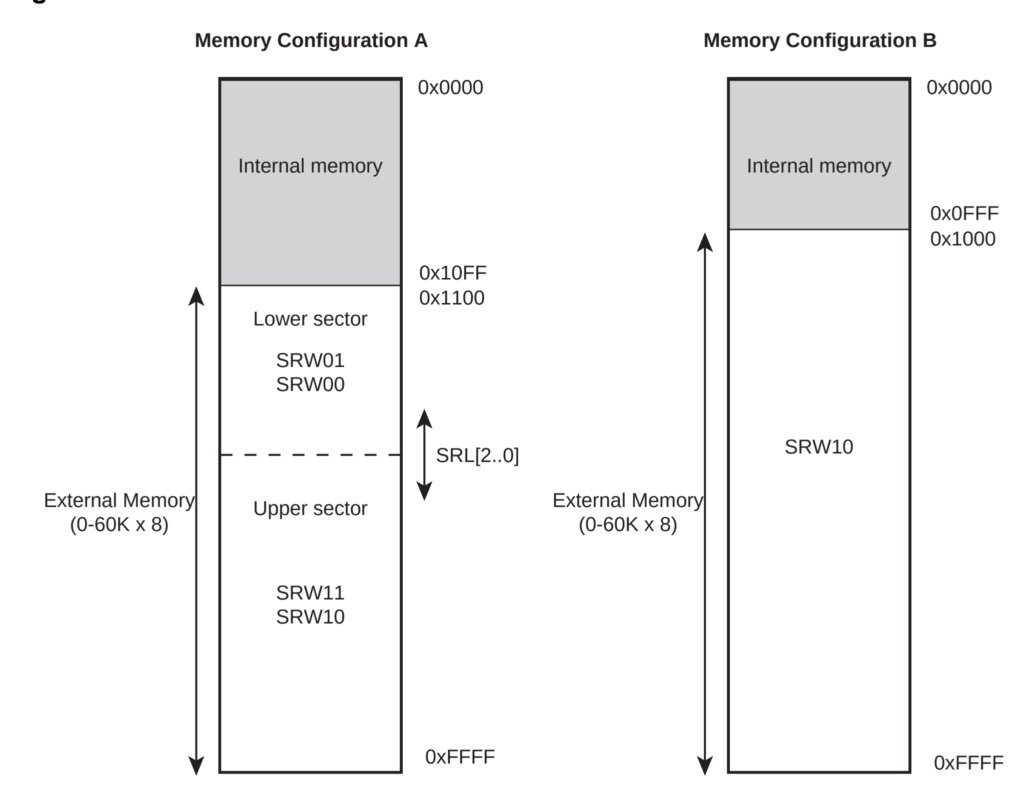
- 四个不同的等待状态设置 (包括无等待状态)。
- 不同的外部存储器可以设置不同的等待状态。
- 地址高字节的位数可以有选择地确定。
- 数据线具有总线保持功能以降低功耗。



综述

使能外部存储器 (XMEM) 时，可以使用专门的外部存储器引脚 （ 参见 P 2Figure 1 ，P 68Table 27 ， P 72Table 33 及 P 79Table 45)。存储器配置如 Figure 11 所示。

Figure 11. 可分区选择的外部存储器



Note: ATmega128 的非 ATmega103 兼容模式：存储器配置 A。
ATmega128 的 ATmega103 兼容模式：存储器配置 B。

ATmega103 兼容性

两个外部存储器控制寄存器 (XMCRA 和 XMCRB) 都位于扩展的 I/O 空间。在 ATmega103 兼容模式下，这些寄存器无法使用，也就无法实现这些寄存器所定义的功能。但是由于这

些功能在 ATmega103 里并不存在，因此与 ATmega103 还是兼容的。 ATmega103 兼容模式带来的限制为：

- 只有两种等待周期选项 (SRW1n = 0b00 和 SRW1n = 0b01)。
- 分配给地址高字节的位数是固定的。
- 外部存储器不能分区，不能有不同的等待周期。
- 没有总线保持功能。
- $\overline{RD}$、 $\overline{WR}$ 和 ALE 引脚只能为输出 (ATmega128 的端口 G)。

使用外部存储器接口

接口包括：

- AD7:0：多工的地址总线和数据总线。
- A15:8：高位地址总线 (位数可配置)。
- ALE：地址锁存使能。
- $\overline{RD}$：读锁存信号。
- $\overline{WR}$：写使能信号。

外部存储器接口控制位于 3 个寄存器当中， MCU 控制寄存器 – MCUCR、外部存储器控制寄存器 A – XMCRA，以及外部存储器控制寄存器 B – XMCRB。

使能 XMEM 接口后， XMEM 接口数据方向寄存器按照接口要求配置。详见 P 61“I/O 端口 ” 。 XMEM 接口将自动检测当前访问的是内部存储器还是外部存储器。如果访问的是外部存储器， XMEM 接口按照 Figure 13 (此图没有等待周期) 输出地址，数据和控制信号。当 ALE 产生由高电平到低电平的变化时， AD7:0 出现有效的地址。数据传输过程中 ALE 保持为低。使能 XMEM 接口之后，即使访问内部存储器也会在地址线，数据线和 ALE 引脚产生动作，但是 $\overline{RD}$ 和 $\overline{WR}$ 信号不会发生变化。禁止外部存储器接口之后，相关引脚就可以使用正常的引脚数据方向设置了。要注意的是，XMEM 接口禁止后内部 SRAM 地址以上的存储器不会映射为内部 SRAM。Figure 12 说明了如何利用一个 8 位锁存器将外部 SRAM 连接到 AVR。

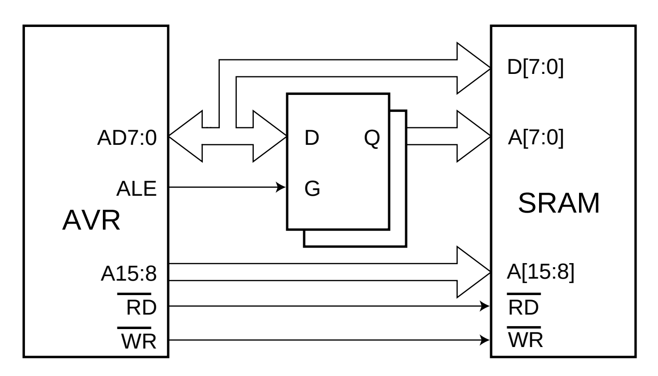
地址锁存要求

由于 XRAM 接口的工作速度很高，当系统的工作条件高于 8 MHz @ 4V 和 4 MHz @ 2.7V 时要注意小心选择地址锁存器。此时典型的老式 74HC 系列锁存器已经无法满足要求了。XRAM接口与74AHC系列的锁存器相兼容。当然，其他满足时序要求的锁存器也是可以使用的。地址锁存器的主要参数为：

- D 到 Q 的传输延迟 (t_{PD})。
- 在 G 拉低之前的数据建立时间 (t_{SU})。
- G 拉低之后的数据 (地址) 保持时间 (t_{TH})。

XRAM 接口的设计出发点是保证 G 拉低之后地址信号保持时间最少为 t_h = 5 ns。具体请参考 Tables 137~Tables 144“ 外部数据存储器时序 ” 中的参数 t_{LAXX_LD}/t_{LLAXX_ST}。计算外部器件的访问时间要求时必须考虑 D 到 Q 的传输延迟 t_{PD}。而 G 拉低之前的数据建立时间 t_{SU}决不能大于地址有效到ALE拉低的时间(t_{AVLLC})减去PCB的线路延迟时间(与负载电容有关)。

Figure 12. 与 AVR 连接的外部 SRAM





上拉和总线保持

PORTx 写 "1“ 时使能 AD7:0 的上拉电阻。为了减少睡眠时的功耗，建议在进入睡眠模式之前对 PORTx 写 "0“ 以禁止上拉电阻。

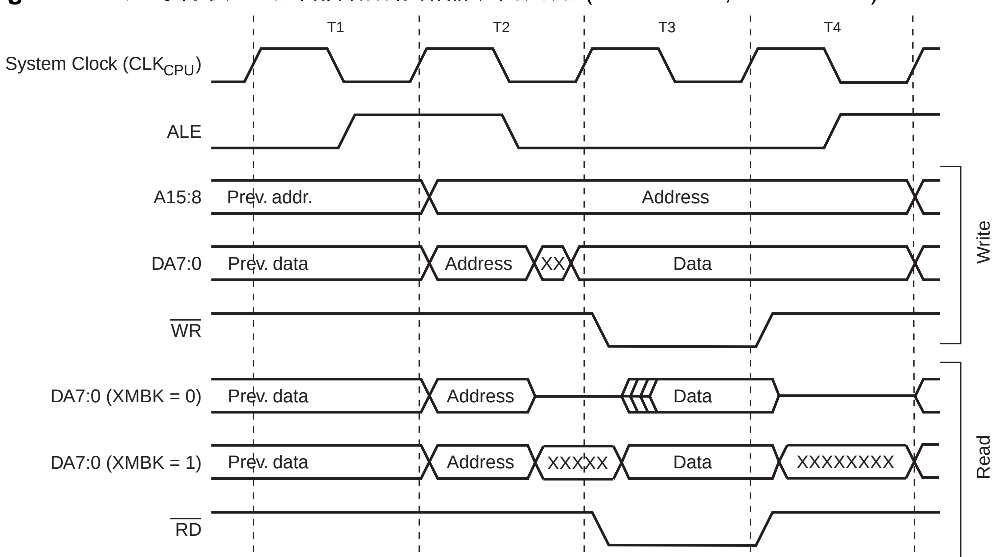
XMEM 接口还提供了 AD7:0 的总线保持功能。这个功能可以通过 P 29“外部存储器控制寄存器 B - XMCRB” 来控制。使能时总线保持器将保持 AD7:0 的前一个数据，同时 XMEM 接口使 AD7:0 总线处于三态。

时序

外部存储器器件有不同的时序要求。为了满足这些要求， ATmega128 XMEM 接口提供了 4 种不同的等待周期，如 Table 4 所示。在选择等待周期之前首先要考虑外部存储器器件的时序要求。相对于 ATmega128 的信号建立要求，最重要的参数是存储器访问时间。外部寄存器的访问时间为接收到片选 / 地址信号直到这个地址的数据出现在总线上的时间间隔。这个访问时间不能大于 ALE 拉低到数据稳定的时间 (参见 Tables 137 到 Tables 144 的 $t_{LLRL} + t_{RLRH} - t_{DVRH}$)。软件可以设置不同的等待周期。而且可以将外部存储器空间分为两个区，每个区具有独立的等待周期。从而可以将两个具有不同时序要求的存储器器件同时连接到 XMEM 接口。具体的 XMEM 接口时序请参见 P 308“ 外部数据存储器时序 ” 的 Table 137 到 Table 144，以及 Figure 156 到 Figure 159。

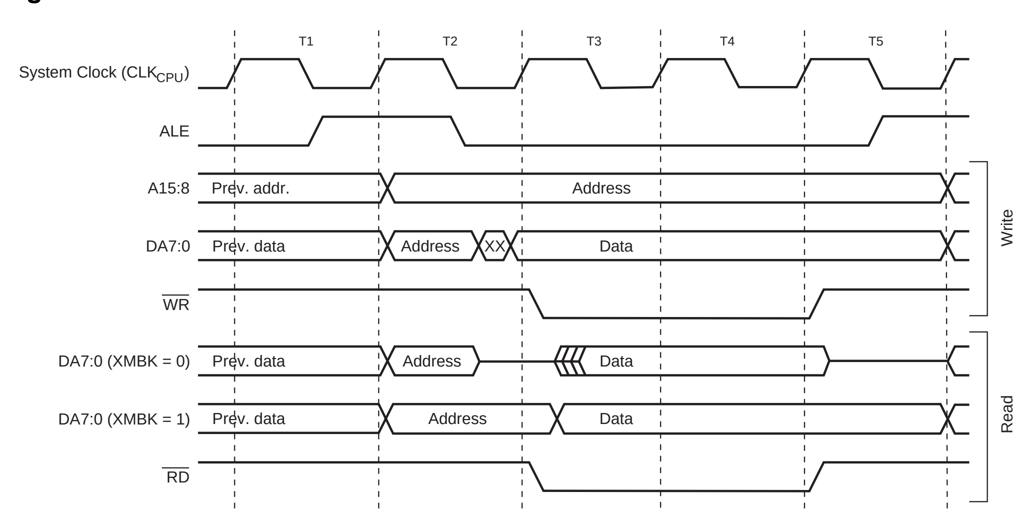
要注意 XMEM 接口是异步的，所有的波形都与内部系统时钟有关。由于内部时钟和外部时钟 (XTAL1) 的相位差与温度和工作电压有关，因此 XMEM 接口不适合于同步操作。

Figure 13. 无等待状态的外部数据存储器访问周期 (SRWn1=0， SRWn0=0)



Note: 1. SRWn1 = SRW11 (高地址存储器区) 或 SRW01 (低地址存储器区) ,SRWn0 = SRW10 (高地址存储器区) 或 SRW00 (低地址存储器区)。
T4 中的 ALE 脉冲仅在下一个指令访问 RAM(内部的或是外部的) 时才会出现。

Figure 14. SRWn1 = 0 及 SRWn0 = 1 时的外部数据存储器访问周期 [1]



Note: 1. SRWn1 = SRW11 (高地址存储器区) 或 SRW01 (低地址存储器区) , SRWn0 = SRW10 (高地址存储器区) 或 SRW00 (低地址存储器区)
T5 中的 ALE 脉冲仅在下一个指令访问 RAM(内部的或是外部的) 时才会出现。

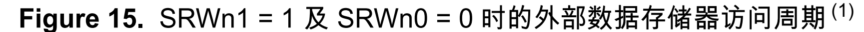
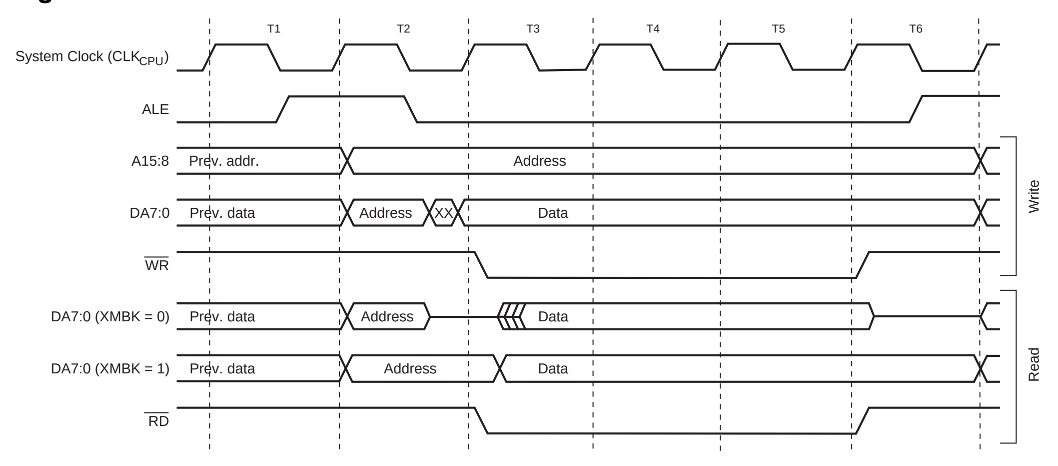


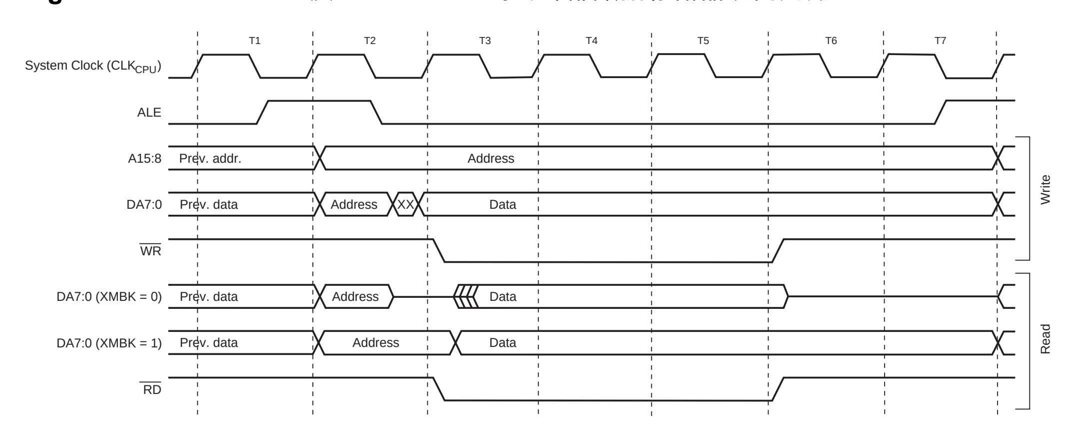
Figure 15. SRWn1 = 1 及 SRWn0 = 0 时的外部数据存储器访问周期 [1]



Note: 1. SRWn1 = SRW11 (高地址存储器区) 或 SRW01 (低地址存储器区) , SRWn0 = SRW10 (高地址存储器区) 或 SRW00 (低地址存储器区)
T6 中的 ALE 脉冲仅在下一个指令访问 RAM(内部的或是外部的) 时才会出现。



Figure 16. SRWn1 = 1 及 SRWn0 = 1 时的外部数据存储器访问周期 [(1)]



Note: 1. SRWn1 = SRW11 (高地址存储器区) 或 SRW01 (低地址存储器区) ,SRWn0 = SRW10 (高地址存储器区) 或 SRW00 (低地址存储器区)
T7 中的 ALE 脉冲仅在下一个指令访问 RAM(内部的或是外部的) 时才会出现。

XMEM 寄存器说明

MCU 控制寄存器 - MCUCR

Bit	7	6	5	4	3	2	1	0	
	SRE	SRW10	SE	SM1	SM0	SM2	IVSEL	IVCE	MCUCR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – SRE: 外部 SRAM/XMEM 使能**

SRE 为 "1“ 时外部存储器接口使能。, 引脚 AD7:0 , A15:8 , ALE , $\overline{WR}$ 和 $\overline{RD}$ 工作于第二功能，且自动按照要求配置端口方向寄存器。SRE 清零将使外部 SRAM 无效，相关端口可以当作普通 I/O 口使用。

- **Bit 6 – SRW10: 等待状态选择位**

对于非 ATmega103 兼容模式，请参见下面的 SRWn 说明 (XMCRA 说明部分)。对于 ATmega103 兼容模式 , SRW10 写 "1“ 将使能等待状态 , 并如 Figure 14 所示在读 / 写过程中插入一个时钟周期。

外部存储器控制寄存器 A - XMCRA

Bit	7	6	5	4	3	2	1	0	
	–	SRL2	SRL1	SRL0	SRW01	SRW00	SRW11	–	XMCRA
读 / 写	R	R/W	R/W	R/W	R/W	R/W	R/W	R	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – Res: 保留**

保留位，读操作返回值为零。在写数据时要写入 0 以保证与未来产品的兼容。

- **Bit 6..4 – SRL2, SRL1, SRL0: 等待状态存储器区限制**

对于不同的外部存储器地址可以配置不同的等待状态。外部存储器地址空间可以分为两个区，而且可以具有独立的等待状态设置位。 SRL2 ， SRL1 和 SRL0 用来对存储器地址

空间进行分区，如 Table 3 和 Figure 11 所示。SRL2，SRL1 和 SRL0 的缺省值为 0，即整个外部存储器地址空间为一个大区。此时等待状态通过 SRW11 和 SRW10 设置。

Table 3. SRL2..0 不同设置对应的分区限制

SRL2	SRL1	SRL0	分区限制
0	0	0	低地址存储器区 = N/A 高地址存储器区 = 0x1100 - 0xFFFF
0	0	1	低地址存储器区 = 0x1100 - 0x1FFF 高地址存储器区 = 0x2000 - 0xFFFF
0	1	0	低地址存储器区 = 0x1100 - 0x3FFF 高地址存储器区 = 0x4000 - 0xFFFF
0	1	1	低地址存储器区 = 0x1100 - 0x5FFF 高地址存储器区 = 0x6000 - 0xFFFF
1	0	0	低地址存储器区 = 0x1100 - 0x7FFF 高地址存储器区 = 0x8000 - 0xFFFF
1	0	1	低地址存储器区 = 0x1100 - 0x9FFF 高地址存储器区 = 0xA000 - 0xFFFF
1	1	0	低地址存储器区 = 0x1100 - 0xBFFF 高地址存储器区 = 0xC000 - 0xFFFF
1	1	1	低地址存储器区 = 0x1100 - 0xDFFF 高地址存储器区 = 0xE000 - 0xFFFF

- **Bit 1 和 MCUCR 的 Bit 6 – SRW11, SRW10: 高地址存储器区等待状态选择位**

SRW11 和 SRW10 用来控制外部存储器高地址区等待状态的数目，如 Table 4 所示。

- **Bit 3..2 – SRW01, SRW00: 低地址存储器区等待状态选择位**

SRW01 和 SRW00 用来控制外部存储器低地址区等待状态的数目，如 Table 4 所示。

Table 4. 等待状态 (1)

SRWn1	SRWn0	等待状态
0	0	无等待周期
0	1	读 / 写操作插入一个等待周期
1	0	读 / 写操作插入两个等待周期
1	1	读 / 写操作插入两个等待周期。输出新地址之前还要插入一个等待周期

Note: 1. n = 0 或 1 (低 / 高地址存储器区)
更进一步的信息请参见 Figures 13~Figures 16，以了解 SRW 如何影响时序。

- **Bit 0 – Res: 保留**

保留位，读操作返回值为零。在写数据时要写入 0 以保证与未来产品的兼容。

外部存储器控制寄存器 B - XMCRB

Bit	7	6	5	4	3	2	1	0	
	XMBK	–	–	–	–	XMM2	XMM1	XMM0	XMCRB
读 / 写	R/W	R	R	R	R	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7– XMBK: 外部存储器总线保持功能使能**

XMBK写"1"将使能AD7:0口线上的总线保持功能。当此功能使能后AD7:0将保持最后的数据不变，即使 XMEM 接口将这些口线设置为三态。 XMBK 为零时总线保持功能禁止。





XMBK不受SRE的限制。因此即使禁止了XMEM接口，只要XMBK为"1“，总线保持功能这样有效。

- **Bit 6..4 – Res: 保留**

保留位，读操作返回值为零。在写数据时要写入 0 以保证与未来产品的兼容。

- **Bit 2..0 – XMM2, XMM1, XMM0: 外部存储器高位地址屏蔽**

在缺省条件下，使能外部存储器之后所有的端口 C 引脚都被用作高位地址。如果系统不需要全部的 60KB 外部存储器地址空间，端口 C 的某些引脚可以释放，用作普通的 I/O，如 Table 5 所示。另外，如 P 32“ 完全使用外部的 64KB 存储器 ” 说明的那样，可以利用 XMMn 来访问外部存储器所有 64KB 的位置。

Table 5. 使能外部存储器时将端口 C 的引脚释放，作为普通口线使用

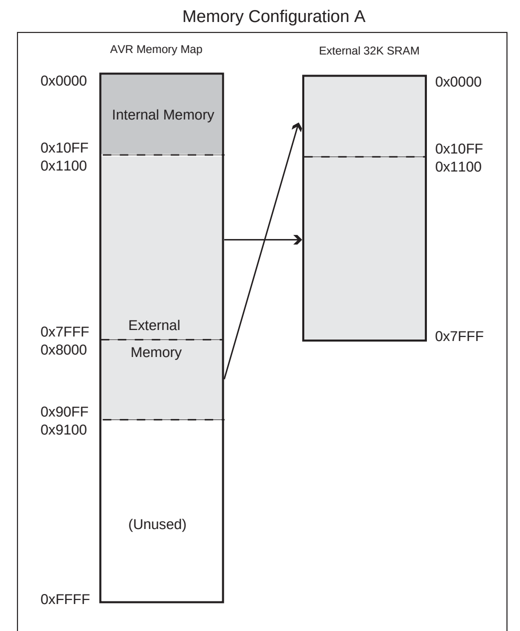
XMM2	XMM1	XMM0	外部存储器地址的位数	可以释放的端口引脚
0	0	0	8 (全部 60 KB 地址空间)	无
0	0	1	7	PC7
0	1	0	6	PC7 - PC6
0	1	1	5	PC7 - PC5
1	0	0	4	PC7 - PC4
1	0	1	3	PC7 - PC3
1	1	0	2	PC7 - PC2
1	1	1	没有高位地址	端口 C 的全部引脚

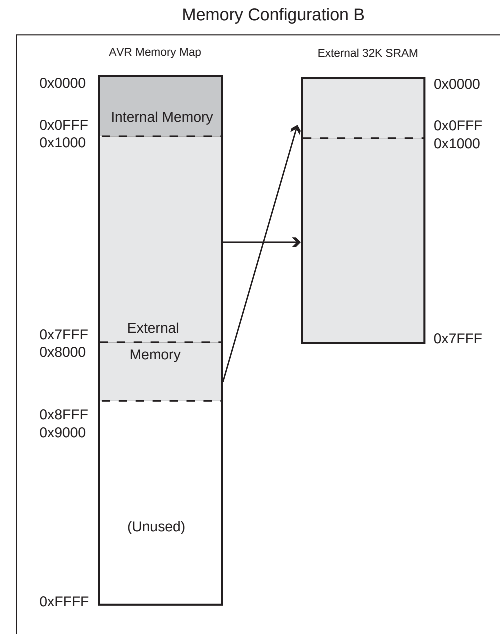
使用外部的存储器 (小于 64KB)

如 Figure 11 所示，外部存储器映射到内部存储器之后，当访问头 4,352 字节时，外部存储器没有定址。就好像外部存储器的头 4,352 字节不可访问 (地址 0x0000 ~ 0x10FF)。但当外部存储器小于 64 KB 时，例如为 32 KB，则通过从 0x8000~0x90FF 的定址，很容易对其进行访问。由于外部存储器地址位 A15 没有与外部存储器相连接，地址 0x8000~0x90FF 将作为外部存储器的 0x0000 ~ 0x10FF。不推荐对 0x90FF 向上寻址，因为这些位置已经被访问。对于应用程序，这 32 KB 外部存储器为从 0x1100~0x90FF 的 32 KB 地址空间，如 Figure 17 所示。存储器配置 B 参见 ATmega103 兼容模式，配置 A 参见非 ATmega103 兼容模式。

当器件设置为 ATmega103 兼容模式。内部地址空间为 4,096 字节。这意味着外部存储器的头 4,096 字节可通过地址 0x8000~0x8FFF 访问。对应用程序，外部 32 KB 存储器是地址为 0x1000~0x8FFF 的线性空间。

Figure 17. 32 KB 外部存储器地址映射





完全使用外部的 64KB 存储器

如 Figure 11 所示，外部存储器映射到内部存储器之后，因此在缺省条件下 MCU 只能访问 60KB 的外部存储器 (地址 0x0000 ~ 0x10FF 为内部存储器所保留)。然而，可以利用屏蔽高位地址的方法来访问整个 64KB 的外部存储器。通过设置端口 C 输出 0x00，并释放最高位，存储器接口就可以访问地址 0x0000 - 0x1FFF 了。请参见如下的例子。

汇编代码例程 [1]

```
; OFFSET 定义为 0x2000 以保证访问的是外部存储器
; 配置端口 C （地址的高字节）输出 0x00，同时释放某些引脚
ldi   r16, 0xFF
out   DDRC, r16
ldi   r16, 0x00
out   PORTC, r16
; 释放 PC7:5
ldi   r16, (1<<XMM1)|(1<<XMM0)
sts   XMCRB, r16
; 写 0xAA 到外部存储器的 0x0001 地址
ldi   r16, 0xaa
sts   0x0001+OFFSET, r16
; 重新使能 PC7:5
ldi   r16, (0<<XMM1)|(0<<XMM0)
sts   XMCRB, r16
; 将 0x55 写入外部存储器地址 (OFFSET + 1)
ldi   r16, 0x55
sts   0x0001+OFFSET, r16
```

C 代码例程 [1]

```
#define OFFSET 0x2000

void XRAM_example(void)
{
unsigned char *p = (unsigned char *) (OFFSET + 1);

DDRC = 0xFF;
PORTC = 0x00;

XMCRB = (1<<XMM1) | (1<<XMM0);

*p = 0xaa;

XMCRB = 0x00;

*p = 0x55;
}
```

Note: 1. 本代码假定已经包含了合适的头文件。

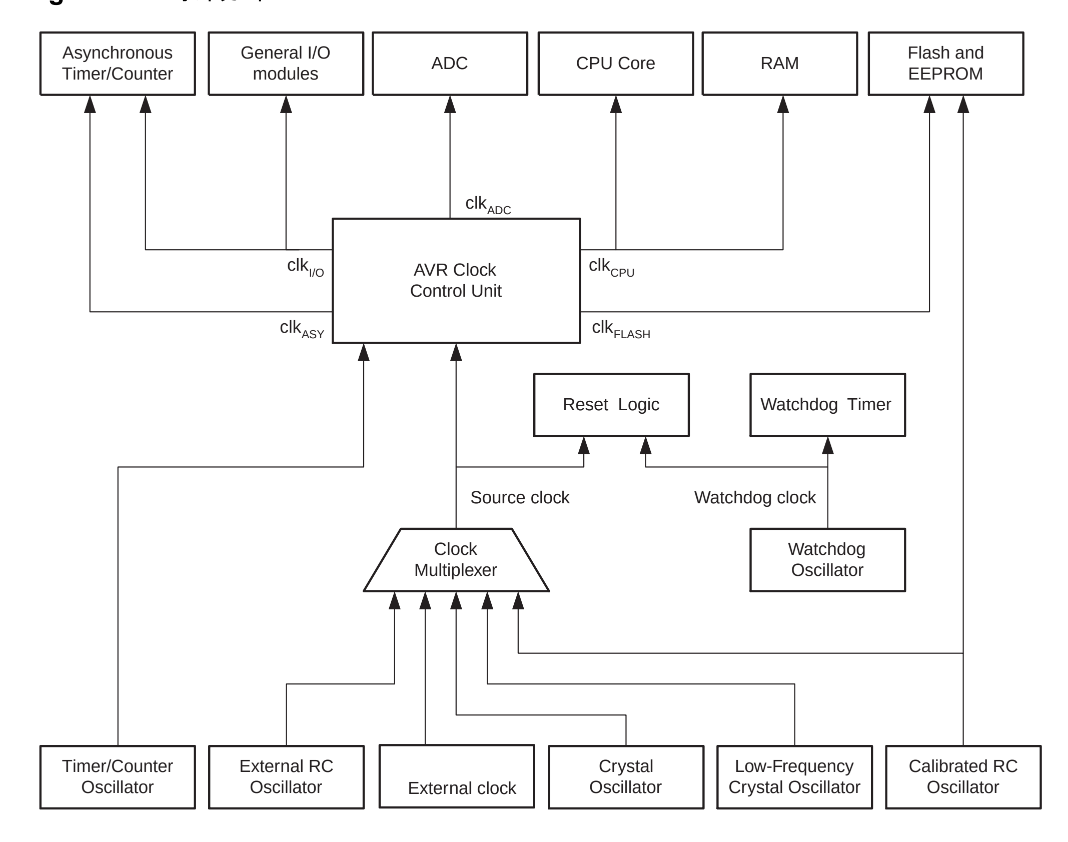
由于这个方法屏蔽了大多数的存储器，要小心使用。

系统时钟及其选项

时钟系统及其分布

Figure 18为AVR的主要时钟系统及其分布。这些时钟并不需要同时工作。为了降低功耗，可以通过使用不同的睡眠模式来禁止无需工作的模块的时钟，如 P 41“ 电源管理及睡眠模式 ” . 所示。

Figure 18. 时钟分布



CPU 时钟 - clk_{CPU}

CPU时钟与操作AVR内核的子系统相连，如通用工作寄存器文件、状态寄存器以及保存堆栈指针的数据存储器。终止 CPU 时钟将使内核停止工作和计算。

I/O 时钟 - $clk_{I/O}$

I/O 时钟用于主要的 I/O 模块，如定时器 / 计数器、SPI 和 USART。I/O 时钟还用于外部中断模块。但是有些外部中断由异步逻辑检测，因此即使 I/O 时钟停止了这些中断仍然可以得到监控。此外，TWI 模块的地址识别功能在没有 $clk_{I/O}$ 的情况下也是异步实现的，使得这个功能在任何睡眠模式下都可以正常工作。

Flash 时钟 - clk_{FLASH}

Flash 时钟控制 Flash 接口的操作。此时钟通常与 CPU 时钟是同步的。

异步定时器时钟 - clk_{ASY}

异步定时器时钟允许异步定时器 / 计数器直接由外部 32 kHz 时钟晶体驱动，使得此定时器 / 计数器即使在睡眠模式下仍然可以为系统提供一个实时时钟。

ADC 时钟 - clk_{ADC}

ADC具有专门的时钟。这样可以在ADC工作的时候停止CPU和I/O时钟以降低数字电路产生的噪声，从而提高 ADC 转换精度。



时钟源

芯片有如下几种通过熔丝位选择的时钟源。时钟输入到 AVR 时钟发生器，并通往其他合适的模块。

Table 6. 时钟源选择

芯片时钟选项	CKSEL3..0[(1)]
外部晶体 / 陶瓷振荡器	1111 - 1010
外部低频晶体	1001
外部 RC 振荡器	1000 - 0101
标定的内部 RC 振荡器	0100 - 0001
外部时钟	0000

Note: 1. 对于所有的熔丝位，“1” 表示未编程，“0” 代表已编程。

每个时钟源在后续部分单独介绍。当 CPU 自掉电模式或省电模式唤醒之后，被选择的时钟源用来为启动过程定时，保证振荡器在开始执行指令之前进入稳定状态。当 CPU 从复位开始工作时，还有额外的延迟时间以保证在开始正常工作之前电源达到稳定电平。看门狗振荡器用来为自己的启动时间定时。看门狗溢出时间所对应的 WDT 振荡器周期数列于 Table 7。看门狗振荡器的频率与工作电压有关，具体请参见 P 313“ATmega128 典型特性”。

Table 7. 看门狗振荡器周期数目

典型的溢出时间 (V_{CC} = 5.0V)	典型的溢出时间 (V_{CC} = 3.0V)	时钟周期数目
4.1 ms	4.3 ms	4K (4,096)
65 ms	69 ms	64K (65,536)

默认时钟源

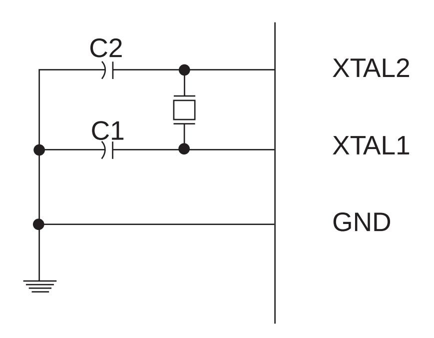
器件出厂时 CKSEL = “0001”， SUT = “10”。该默认时钟源为最长启动时间的内部 RC 振荡器。该默认设置保证所有的用户通过系统内或并行编程得到其需要的时钟源。

晶体振荡器

XTAL1 和 XTAL2 分别为用作片内振荡器的反向放大器的输入和输出，如 Figure 19 所示。这个振荡器可以使用石英晶体，也可以使用陶瓷谐振器。熔丝位 CKOPT 用来选择这两种放大器模式的其中之一。当 CKOPT 被编程时振荡器在输出引脚产生满幅度的振荡。这种模式适合于噪声环境，以及需要通过 XTAL2 驱动第二个时钟缓冲器的情况。而且这种模式的频率范围比较宽。当保持 CKOPT 为未编程状态时，振荡器的输出信号幅度比较小。其优点是大大降低了功耗，但是频率范围比较窄，而且不能驱动其他时钟缓冲器。

对于谐振器， CKOPT 未编程时的最大频率为 8 MHz， CKOPT 编程时为 16 MHz。 C1 和 C2 的数值要一样，不管使用的是晶体还是谐振器。最佳的数值与使用的晶体或谐振器有关，还与杂散电容和环境的电磁噪声有关。 Table 8 给出了针对晶体选择电容的一些指南。对于陶瓷谐振器，应该使用厂商提供的数值。若想得到更多的有关如何选择电容以及振荡器如何工作的信息，请参考多用途振荡器应用手册。

Figure 19. 晶体振荡器连接图



振荡器可以工作于三种不同的模式，每一种都有一个优化的频率范围。工作模式通过熔丝位 CKSEL3..1 来选择，如 Table 8 所示。

Table 8. 晶体振荡器工作模式

CKOPT	CKSEL3..1	频率范围 (MHz)	使用晶体时电容 C1 和 C2 的推荐范围
1	101(1)	0.4 - 0.9	–
1	110	0.9 - 3.0	12 pF - 22 pF
1	111	3.0 - 8.0	12 pF - 22 pF
0	101, 110, 111	1.0 -	12 pF - 22 pF

Note: 1. 此选项不适用于晶体，只能用于陶瓷谐振器。

如 Table 9 所示，熔丝位 CKSEL0 以及 SUT1..0 用于选择启动时间。

Table 9. 晶体振荡器时钟选项对应的启动时间

CKSEL0	SUT1..0	掉电模式和省电模式的启动时间	复位时的额外延迟时间 (V_{CC} = 5.0V)	推荐用法
0	00	258 CK[1]	4.1 ms	陶瓷谐振器，电源快速上升
0	01	258 CK[1]	65 ms	陶瓷谐振器，电源缓慢上升
0	10	1K CK[2]	–	陶瓷谐振器， BOD 使能
0	11	1K CK[2]	4.1 ms	陶瓷谐振器，电源快速上升
1	00	1K CK[2]	65 ms	陶瓷谐振器，电源缓慢上升
1	01	16K CK	–	晶体振荡器， BOD 使能
1	10	16K CK	4.1 ms	晶体振荡器，电源快速上升
1	11	16K CK	65 ms	晶体振荡器，电源缓慢上升

Notes: 1. 这些选项只能用于工作频率不太接近于最大频率，而且启动时的频率稳定性对于应用而言不重要的情况。不适用于晶体。
2. 这些选项是为陶瓷谐振器设计的，可以保证启动时频率足够稳定。而且当工作频率不太接近于最大频率，而且启动时的频率稳定性对于应用而言不重要时也可以适用于晶体。

低频晶体振荡器

为了使用 32.768 kHz 钟表晶体作为器件的时钟源，必须将熔丝位 CKSEL 设置为 “1001” 以选择低频晶体振荡器。晶体的连接方式如 Figure 19 所示。通过对熔丝位 CKOPT 的编程，用户可以使能 XTAL1 和 XTAL2 的内部电容，从而去除外部电容。内部电容的标称数值为 36 pF。

选择了这个振荡器之后，启动时间由熔丝位 SUT 确定，如 Table 10 所示。

Table 10. 低频晶体振荡器的启动时间

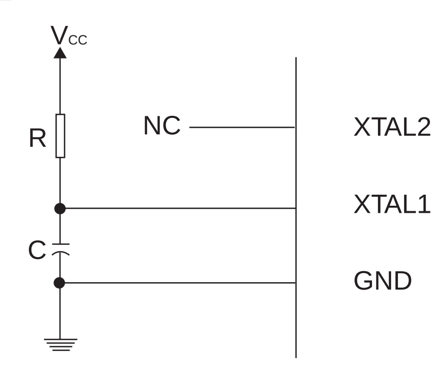
SUT1..0	掉电模式和省电模式的启动时间	复位时的额外延迟时间 (V_{CC} = 5.0V)	推荐用法
00	1K CK[1]	4.1 ms	电源快速上升，或是 BOD 使能
01	1K CK[1]	65 ms	电源缓慢上升
10	32K CK	65 ms	启动时频率已经稳定
11	保留		

Note: 1. 这些选项只能用于启动时的频率稳定性对应用而言不重要的情况。

外部 RC 振荡器

对于时间不敏感的应用可以使用 Figure 20 的外部 RC 振荡器。频率可以通过方程 $f = 1/(3RC)$ 进行粗略地估计。电容 C 至少要 22 pF。通过编程熔丝位 CKOPT，用户可以使能 XTAL1和GND之间的片内 36 pF电容，从而无需外部电容。若想获取有关振荡器如何工作以及如何选择 R 和 C 的具体信息，请参考外部 RC 振荡器应用手册。

Figure 20. 外部 RC 配置



振荡器可以工作于四个不同的模式，每个模式有自己的优化频率范围。工作模式通过熔丝位 CKSEL3..0 选取，如 Table 11 所示。

Table 11. 外部 RC 振荡器工作模式

CKSEL3..0	频率范围 (MHz)
0101	- 0.9
0110	0.9 - 3.0
0111	3.0 - 8.0
1000	8.0 - 12.0

选择了这个振荡器之后，启动时间由熔丝位 SUT 确定，如 Table 12 所示。

Table 12. 外部 RC 振荡器的启动时间

SUT1..0	掉电模式和省电模式的启动时间	复位时的额外延迟时间 (V_{CC} = 5.0V)	推荐用法
00	18 CK	–	BOD 使能
01	18 CK	4.1 ms	电源快速上升
10	18 CK	65 ms	电源缓慢上升
11	6 CK(1)	4.1 ms	电源快速上升，或是 BOD 使能

Note: 1. 这些选项只能用于工作频率不太接近于最大频率时的情况。

标定的片内 RC 振荡器

标定的片内 RC 振荡器提供了固定的 1.0、 2.0、 4.0 或 8.0 MHz 的时钟。这些频率都是 5V、 25°C 下的标称数值。这个时钟也可以作为系统时钟，只要按照 Table 13 对熔丝位 CKSEL进行编程即可。选择这个时钟(此时不能对CKOPT进行编程)之后就无需外部器件了。复位时硬件将标定字节加载到 OSCCAL 寄存器，自动完成对 RC 振荡器的标定。在 5V，25°C 和频率为 1.0 MHz 时，这种标定可以提供标称频率 ± 3% 的精度；使用 www.atmel.com/avr 中所给出的方法，可在任何电压、任何温度下，使精度达到 ± 1% 。当使用这个振荡器作为系统时钟时，看门狗仍然使用自己的看门狗定时器作为溢出复位的依据。更多的有关标定数据的信息请参见 P 270“ 标定字节 ” 。

Table 13. 片内标定的 RC 振荡器工作模式

CKSEL3..0	标称频率 (MHz)
0001(1)	1.0
0010	2.0
0011	4.0
0100	8.0

Note: 1. 出厂时的设置。

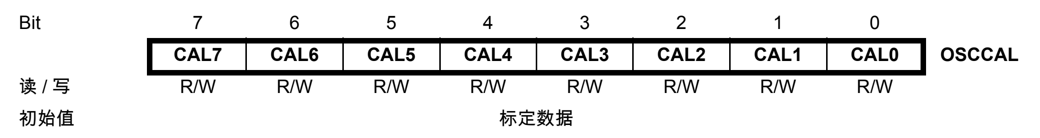
选择了这个振荡器之后，启动时间由熔丝位 SUT 确定，如 Table 14 所示。 XTAL1 和 XTAL2 要保持为空 (NC)。

Table 14. 内部标定 RC 振荡器的启动时间

SUT1..0	掉电模式和省电模式的启动时间	复位时的额外延迟时间 (V_{CC} = 5.0V)	推荐用法
00	6 CK	–	BOD 使能
01	6 CK	4.1 ms	电源快速上升
10(1)	6 CK	65 ms	电源缓慢上升
11	保留		

Note: 1. 出厂时的设置。

振荡器标定寄存器 - OSCCAL



Bit	7	6	5	4	3	2	1	0	
	CAL7	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	OSCCAL
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	标定数据								

Note: ATmega103 兼容模式没有 OSCCAL 寄存器。

- **Bits 7..0 – CAL7..0: 振荡器标定数据**

将标定数据写入这个地址可以对内部振荡器进行调节以消除由于生产工艺所带来的振荡器频率偏差。复位时 1 MHz 的标定数据 (标识数据的高字节，地址为 0x00) 自动加载到 OSCCAL 寄存器。如果需要内部 RC 振荡器工作于其他频率，标定数据必须人工加载：首先通过编程器读取标识数据，然后将标定数据保存到 Flash 或 EEPROM 之中。这些数据可以通过软件读取，然后加载到 OSCCAL 寄存器。当 OSCCAL 为零时振荡器以最低频率工作。当对其写如不为零的数据时内部振荡器的频率将增长。写入 $FF 即得到最高频率。标定的振荡器用来为访问 EEPROM 和 Flash 定时。有写 EEPROM 和 Flash 的操作

时不要将频率标定到超过标称频率的 10%，否则写操作有可能失败。要注意振荡器只对 1.0、2.0、4.0 和 8.0 MHz 这四种频率进行了标定，其他频率则无法保证，见 Table 15。.

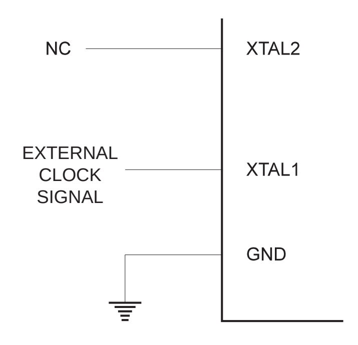
Table 15. 内部 RC 振荡器频率范围

OSCCAL 数值	最小频率，标称频率的百分比 (%)	最大频率，标称频率的百分比 (%)
$00	50	100
$7F	75	150
$FF	100	200

外部时钟

为了从外部时钟源驱动芯片， XTAL1 必须如 Figure 21 所示的进行连接。同时，熔丝位 CKSEL 必须编程为 “0000”。若熔丝位 CKOPT 也被编程，用户就可以使用内部的 XTAL1 和 GND 之间的 36 pF 电容。

Figure 21. 外部时钟驱动配置图



选择了这个振荡器之后，启动时间由熔丝位 SUT 确定，如 Table 16 所示。

Table 16. 外部时钟的启动时间

SUT1..0	掉电模式和省电模式的启动时间	复位时的额外延迟时间 (V_{CC} = 5.0V)	推荐用法
00	6 CK	–	BOD 使能
01	6 CK	4.1 ms	电源快速上升
10	6 CK	65 ms	电源缓慢上升
11	保留		

为了保证 MCU 能够稳定工作，不能突然改变外部时钟源的振荡频率。工作频率突变超过 2% 将会产生异常现象。应该在 MCU 保持复位状态时改变外部时钟的振荡频率。



定时器 / 计时器振荡器

对于拥有定时器 / 振荡器引脚 (TOSC1 和 TOSC2) 的 AVR 微处理器，晶体可以直接与这两个引脚连接，无需外部电容。此振荡器针对 32.768 kHz 的钟表晶体作了优化。不建议在 TOSC1 引脚输入振荡信号。

XTAL 分频控制寄存器 - XDIV

XTAL 分频控制寄存器用于源时钟的分频 , 分频范围为 2 - 129。这个特性可以用来降低功耗。

Bit	7	6	5	4	3	2	1	0	
	XDIVEN	XDIV6	XDIV5	XDIV4	XDIV3	XDIV2	XDIV1	XDIV0	XDIV
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – XDIVEN: XTAL 分频使能**

XDIVEN 为 "1“ 时 CPU 和其他外设的时钟频率 ($clk_{I/O}$、clk_{ADC}、clk_{CPU}、clk_{FLASH}) 以 XDIV6 - XDIV0 确定的数值为因子进行分频。这一位可以在运行的过程当中进行修改。

- **Bits 6..0 – XDIV6..XDIV0: XTAL 分频选择位 6 - 0**

XDIV6 - XDIV0 定义了系统使用的分频因子。假定其数值以 d 表示 , 分频之后的 CPU 和其他外设的时钟频率 f_{CLK} 计算公式为 ：

$$f_{CLK} = \frac{\text{Source clock}}{129 - d}$$

只有在XDIVEN为零时才可以修改这些数据。对XDIVEN写"1“时 , 同时写入XDIV6..XDIV0 的数据立即生效。否则这些数据无效。由于分频器分频的是 输入到 MCU 的主时钟 , 外设的时钟同时也被分频。

Note: 系统时钟被分频时不应该使用定时器/计数器0。因为它同时用作异步定时器/计数器。即使它工作于同步模式，其时钟也是不会被分频的。若使用这个定时器 / 计数器，后果是中断可能丢失，访问其定时器也可能失败。

电源管理及睡眠模式

睡眠模式可以使应用程序关闭 MCU 中没有使用的模块，从而降低功耗。 AVR 具有不同的睡眠模式，允许用户根据自己的应用要求实施剪裁。

进入睡眠模式的条件是置位寄存器 MCUCR 的 SE，然后执行 SLEEP 指令。具体哪一种模式 (空闲模式、ADC 噪声抑制模式、掉电模式、省电模式、Standby 模式和扩展 Standby 模式) 由 MCUCR 的 SM2、 SM1 和 SM0 决定，如 Table 17 所示。使能的中断可以将进入睡眠模式的 MCU 唤醒。经过启动时间，外加 4 个时钟周期后， MCU 就可以运行中断例程了。然后返回到 SLEEP 的下一条指令。唤醒时不会改变寄存器文件和 SRAM 的内容。如果在睡眠过程中发生了复位，则 MCU 唤醒后从中断向量开始执行。

P 33Figure 18 介绍了ATmega128不同的时钟系统及其分布。此图在选择合适的睡眠模式时非常有用。

MCU 控制寄存器 - MCUCR

MCU 控制寄存器包含了电源管理的控制位。

Bit	7	6	5	4	3	2	1	0	
	SRE	SRW10	SE	SM1	SM0	SM2	IVSEL	IVCE	MCUCR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 5 – SE: 睡眠使能**

为了使 MCU 在执行 SLEEP 指令后进入睡眠模式， SE 必须置位。为了确保进入睡眠模式是程序员的有意行为，建议仅在 SLEEP 指令的前一条指令置位 SE。一旦唤醒立即清除 SE。

- **Bits 4..2 – SM2..0: 睡眠模式选择位**

如 Table 17 所示，这些位用于选择具体的睡眠模式。

Table 17. 睡眠模式选择

SM2	SM1	SM0	睡眠模式
0	0	0	空闲模式
0	0	1	ADC 噪声抑制模式
0	1	0	掉电模式
0	1	1	省电模式
1	0	0	保留
1	0	1	保留
1	1	0	Standby 模式 (1)
1	1	1	扩展的 Standby 模式 (1)

Note: 1. 仅在使用外部晶体或谐振器时 Standby 模式和扩展的 Standby 模式才可用。



空闲模式

当 SM2..0 为 000 时，SLEEP 指令将使 MCU 进入空闲模式。在此模式下，CPU 停止运行，而 SPI、 USART、模拟比较器、 ADC、两线接口、定时器 / 计数器、看门狗和中断系统继续工作。这个睡眠模式只停止了 clk_{CPU} 和 clk_{FLASH}，其他时钟则继续工作。

内外部中断都可以唤醒 MCU。如果不需要从模拟比较器中断唤醒 MCU，为了减少功耗，可以切断比较器的电源。方法是置位模拟比较器控制和状态寄存器 ACSR 的位 ACD。如果 ADC 使能，进入此模式后将自动启动一次转换。

ADC 噪声抑制模式

当 SM2..0 为 001 时，SLEEP 指令将使 MCU 进入噪声抑制模式。在此模式下，CPU 停止运行，而 ADC、外部中断、两线接口地址配置、定时器 / 计数器 0 和看门狗继续工作。这个睡眠模式只停止了 $clk_{I/O}$、 clk_{CPU} 和 clk_{FLASH}，其他时钟则继续工作。

此模式提高了 ADC 的噪声环境，使得转换精度更高。 ADC 使能的时候，进入此模式将自动启动一次 AD 转换。ADC 转换结束中断、外部复位、看门狗复位、BOD 复位、两线接口地址匹配中断、定时器 / 计数器 0 中断、 SPM/EEPROM 准备好中断、外部中断 INT7:4，或外部中断 INT3:0 可以将 MCU 从 ADC 噪声抑制模式唤醒。

掉电模式

当 SM2..0 为 010 时， SLEEP 指令将使 MCU 进入掉电模式。在此模式下，外部晶体停振，而外部中断、两线接口地址匹配及看门狗 （如果使能的话）继续工作。只有外部复位、看门狗复位、 BOD 复位、两线接口地址匹配中断、外部电平中断 INT7:4，或外部中断 INT3:0 可以使 MCU 脱离掉电模式。这个睡眠模式停止了所有的时钟，只有异步模块可以继续工作。

当使用外部电平中断方式将 MCU 从掉电模式唤醒时，必须保持外部电平一定的时间。具体请参见 P 84“ 外部中断 ” 。

从施加掉电唤醒条件到真正唤醒有一个延迟时间，此时间用于时钟重新启动并稳定下来。唤醒周期与由熔丝位 CKSEL 定义的复位周期是一样的，如 P 34“ 时钟源 ” 。

省电模式

当 SM2..0 为 011 时， SLEEP 指令将使 MCU 进入省电模式。这一模式与掉电模式只有一点不同：

如果定时器 / 计数器 0 为异步驱动 ,即寄存器 ASSR 的 AS0 置位 ,则定时器 / 计数器 0 在睡眠时继续运行。除了掉电模式的唤醒方式，定时器 / 计数器 0 的溢出中断和比较匹配中断也可以将 MCU 从休眠方式唤醒，只要 TIMSK 使能了这些中断，而且 SREG 的全局中断使能位 I 置位。

如果异步定时器不是异步驱动的，建议使用掉电模式，而不是省电模式。因为在省电模式下，若 AS0 为 0，则 MCU 唤醒后异步定时器的寄存器数值是没有定义的。

这个睡眠模式停止了除 clk_{ASY} 以外所有的时钟，只有异步模块可以继续工作。

Standby 模式

当 SM2..0 为 110 时， SLEEP 指令将使 MCU 进入 Standby 模式。这一模式与掉电模式唯一的不同之处在于振荡器继续工作。其唤醒时间只需要 6 个时钟周期。

扩展 Standby 模式

当 SM2..0 为 111 时， SLEEP 指令将使 MCU 进入扩展的 Standby 模式。这一模式与省掉电模式唯一的不同之处在于振荡器继续工作。其唤醒时间只需要 6 个时钟周期。

Table 18. 在不同睡眠模式下活动的时钟以及唤醒源

	工作的时钟					振荡器		唤醒源					
睡眠模式	clk$_{CPU}$	clk$_{FLASH}$	clk$_{IO}$	clk$_{ADC}$	clk$_{ASY}$	使能的主时钟	使能的定时器时钟	INT7:0	TWI 地址匹配	定时器 0	SPM/E^2 准备好	ADC	其他 I/O
空闲模式			X	X	X	X	X[2]	X	X	X	X	X	X
ADC 噪声抑制模式				X	X	X	X[2]	X[3]	X	X	X	X	
掉电模式								X[3]	X				
省电模式					X[2]		X[2]	X[3]	X	X[2]			
Standby 模式 [1]						X		X[3]	X				
扩展的 Standby 模式 [1]					X[2]	X	X[2]	X[3]	X	X[2]			

Notes:
1. 时钟源为外部晶体或谐振器。
2. ASSR 的 AS0 置位。
3. INT3:0 或电平中断 INT7:4。

最小化功耗

试图降低 AVR 控制系统的功耗时需要考虑几个问题。一般来说，要尽可能利用睡眠模式，并且使尽可能少的模块继续工作。不需要的功能必须禁止。下面的模块需要特殊考虑以达到尽可能低的功耗。

模数转换器

使能时， ADC 在睡眠模式下继续工作。为了降低功耗，在进入睡眠模式之前需要禁止 ADC。重新启动后的第一次转换为扩展的转换。具体请参照 P 213“ 模数转换器 ”。

模拟比较器

在空闲模式时，如果没有使用模拟比较器，可以将其关闭。在 ADC 噪声抑制模式下也是如此。在其他睡眠模式模拟比较器是自动关闭的。如果模拟比较器使用了内部电压基准源，则不论在什么睡眠模式下都需要关闭它。否则内部电压基准源将一直使能。请参见 P 210“ 模拟比较器 ” 以了解如何配置模拟比较器。

掉电检测器

如果应用没有利用掉电检测器 BOD，这个模块也可以关闭。如果编程熔丝位 BODEN 使能 BOD 功能，它将在各种睡眠模式下继续工作，从而消耗电流。在深层次的睡眠模式下，这个电流将占总电流的很大比重。请参看 P 44“ 掉电检测器 ” 以了解如何配置 BOD。

片内基准电压

当使用 BOD、模拟比较器和 ADC 时可能需要内部电压基准源。若这些模块都禁止了，则基准源也可以禁止。重新使能后用户必须等待基准源稳定之后才可以使用。如果基准源在睡眠过程中是使能的，其输出立即可以使用。请参见 P 50“ 片内基准电压 ” 以了解基准源启动时间的细节。

看门狗定时器

如果应用没有利用看门狗，这个模块就可以关闭。若使能，则在任何睡眠模式下都持续工作，从而消耗电流。在深层次的睡眠模式下，这个电流将占总电流的很大比重。请参看 P 51“ 看门狗定时器 ” 以了解如何配置看门狗定时器。

端口引脚

进入睡眠模式时，所有的端口引脚都应该配置为只消耗最小的功耗。最重要的是避免驱动电阻性负载。在睡眠模式下 I/O 时钟 $clk_{I/O}$ 和 ADC 时钟 clk_{ADC} 都被停止了，输入缓冲器也禁止了。从而保证输入电路不会消耗电流。某些输入逻辑是使能的，用来检测唤醒条件。具体的引脚请参见 P 65“ 数字输入使能和睡眠模式 ”。此时输入不能悬空，信号电平也不应该接近 $V_{CC}/2$，否则输入缓冲器会消耗过多的电流。

JTAG 接口与片内调试系统

当通过设置 OCDEN 熔丝位使能片内调试系统，且进入掉电或省电模式时，主时钟源仍然使能。此时仍会增加电流消耗。有三种方式可以避免该情况的发生：

- 禁止 OCDEN 熔丝位。
- 禁止 JTAGEN 熔丝位。
- 对 MCUCSR 寄存器的 JTD 位写 1。

当 JTAG 接口使能而 JTAG TAP 控制器没有移出数据， TDO 引脚悬空。若与 TDO 引脚的硬件连接没有拉高逻辑电平，功耗将增加。注意，在以后的芯片中扫描链中的 TDI 引脚含有上拉电阻避免该问题。对 MCUCSR 寄存器的 JTD 位写 1 或不对 JTAG 熔丝位编程禁用 JTAG 接口。

系统控制和复位

复位 AVR

复位时所有的 I/O 寄存器都被设置为初始值，程序从复位向量处开始执行。复位向量处的指令必须是绝对跳转 JMP 指令，以使程序跳转到复位处理例程。如果程序永远不会使能中断，则中断向量可以由一般的程序代码所覆盖。Figure 22 为复位逻辑的电路图。Table 19 则定义了复位电路的电气参数。

复位源生效时 I/O 端口立即复位为初始值，不需要任何时钟的辅助。

当所有的复位信号消失之后，延迟计数器被激活，从而延长了内部复位，并使得在 MCU 正常工作之前电源达到稳定的电平。延迟计数器的溢出时间通过熔丝位 CKSEL 由用户设定。延迟时间的选择请参见 P 34“ 时钟源 ” 。

复位源

ATmega128 有 5 个复位源：

- 上电复位。当电源电压低于上电复位门限 (V_{POT}) 时， MCU 复位。
- 外部复位。当引脚 $\overline{\text{RESET}}$ 上的低电平持续时间大于最小脉冲宽度时 MCU 复位。
- 看门狗复位。当看门狗使能并且看门狗定时器超时时复位发生。
- 掉电检测复位。当掉电检测复位功能使能，切电源电压低于掉电检测复位门限 (V_{BOT}) 时 MCU 即复位。
- JTAG AVR 复位：当复位寄存器为 1 时 MCU 即复位。参见 P 235“IEEE 1149.1 (JTAG) 边界扫描 ” 。

Figure 22. 复位逻辑

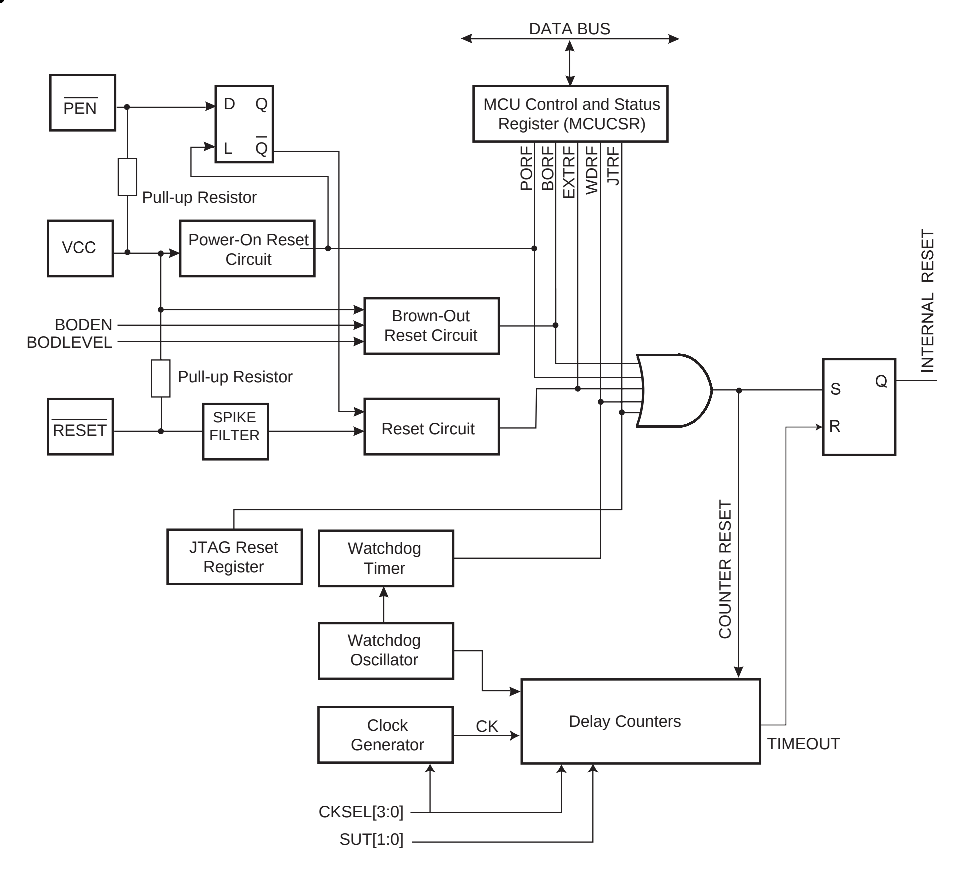


Table 19. 复位特性

符号	参数	条件	最小值	典型值	最大值	单位
V_{POT}	上电复位门限电压 (上升沿)			1.4	2.3	V
	上电复位门限电压 (下降沿)(1)			1.3	2.3	V
V_{RST}	$\overline{\text{RESET}}$ 门限电压		0.2 V_{CC}		0.85 V_{CC}	V
t_{RST}	$\overline{\text{RESET}}$ 最小脉冲宽度		1.5			µs
V_{BOT}	掉电检测复位门限电压 (2)	BODLEVEL = 1	2.4	2.6	2.9	V
		BODLEVEL = 0	3.7	4.0	4.5	
t_{BOD}	触发掉电检测复位的低电平的最小持续时间	BODLEVEL = 1		2		µs
		BODLEVEL = 0		2		µs
V_{HYST}	掉电检测器的容限			100		mV

Notes: 1. 电压下降时，只有电压低于 V_{POT} 时复位才会发生。
2. 一些器件的 V_{BOT} 可能比标称的最小工作电压还要低。这些器件在生产测试过程中进行了 $V_{CC} = V_{BOT}$ 的测试，保证在 V_{CC} 下降到处理器无法正常工作之前产生掉电检测复位。ATmega128L 的测试条件为 BODLEVEL=1，ATmega128 的测试条件为 BODLEVEL=0。BODLEVEL=1 不适用于 ATmega128。

上电复位

上电复位 (POR) 脉冲由片内检测电路产生。检测电平列于 Table 19。 POR 在 V_{CC} 低于检测电平时产生。 POR 电路可以用来触发启动复位，或者用来检测电源故障。

POR 电路保证器件在上电时复位。V_{CC} 达到上电门限电压后触发延迟计数器。在计数器溢出之前器件一直保持为复位状态。当 V_{CC} 下降时，只要低于检测门限，RESET 信号立即生效。

Figure 23. MCU 启动过程，$\overline{\text{RESET}}$ 连接到 V_{CC}.

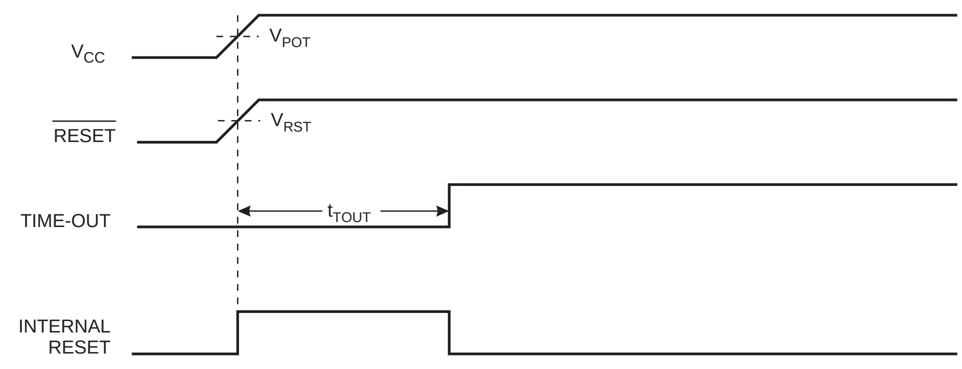
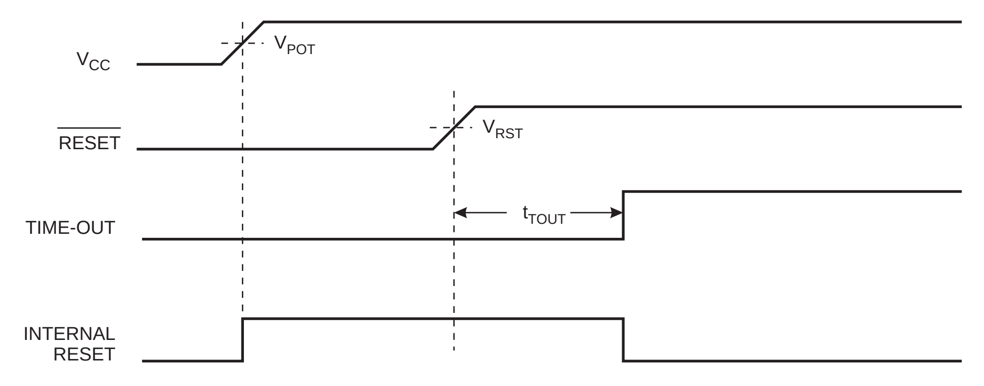




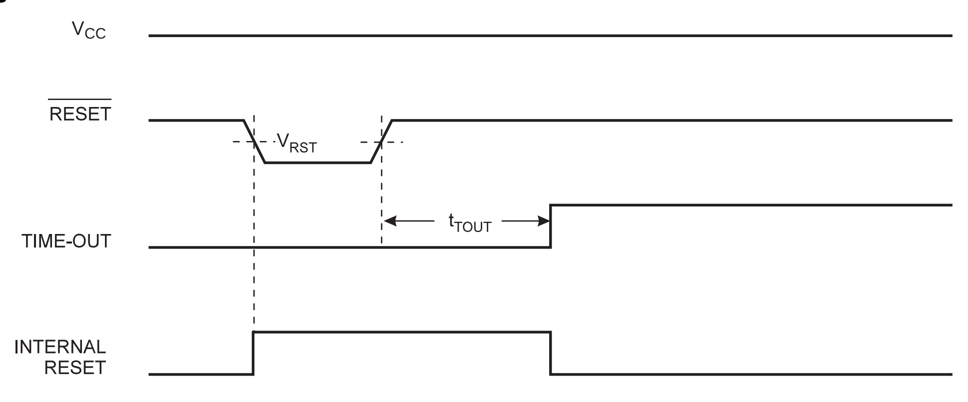
Figure 24. MCU 启动过程，$\overline{\text{RESET}}$ 由外电路控制



外部复位

外部复位由外加于 $\overline{\text{RESET}}$ 引脚的低电平产生。当复位低电平持续时间大于最小脉冲宽度时 (参见 Table 19) 即触发复位过程，即使此时并没有时钟信号在运行。当外加信号达到复位门限电压 V_{RST}(上升沿) 时，t_{TOUT} 延时周期开始。延时结束后 MCU 即启动。

Figure 25. 工作过程中发生外部复位



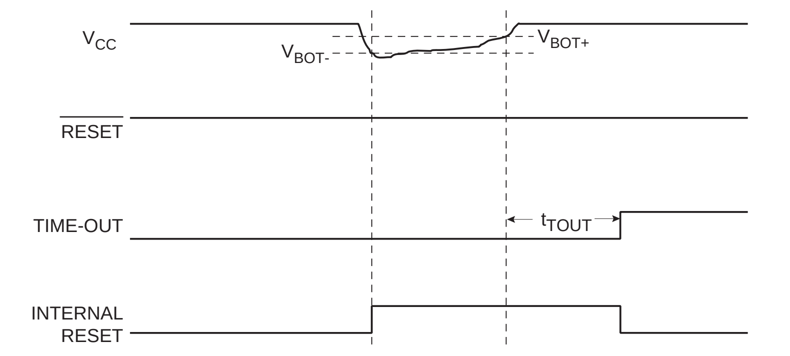
掉电检测复位

ATmega128 具有片内 BOD(Brown-out Detection) 电路，通过与固定的触发电平的对比来检测工作过程中 V_{CC} 的变化。此触发电平可以通过设置熔丝位 BODLEVEL 来设定 BOD 电平为 2.7V (BODLEVEL 不编程) 或 4.0V (BODLEVEL 被编程)。触发电平还具有迟滞功能以消除电源尖峰的影响。这个迟滞功能可以解释为 $V_{BOT+} = V_{BOT} + V_{HYST}/2$ 以及 $V_{BOT-} = V_{BOT} - V_{HYST}/2$。

BOD 电路的开关由熔丝位BODEN控制。当BOD使能后(BODEN被编程)，一旦V_{CC}下降到触发电平以下 (V_{BOT-}，Figure 26)， BOD 复位立即被激发。当 V_{CC} 上升到触发电平以上时(V_{BOT+}，Figure 26)，延时计数器开始计数，一旦超过溢出时间t_{TOUT}，MCU即恢复工作。

如果 V_{CC} 一直低于触发电平并保持如 Table 19 所示的时间 t_{BOD}， BOD 电路将只检测电压跌落。

Figure 26. 工作过程中发生掉电检测复位

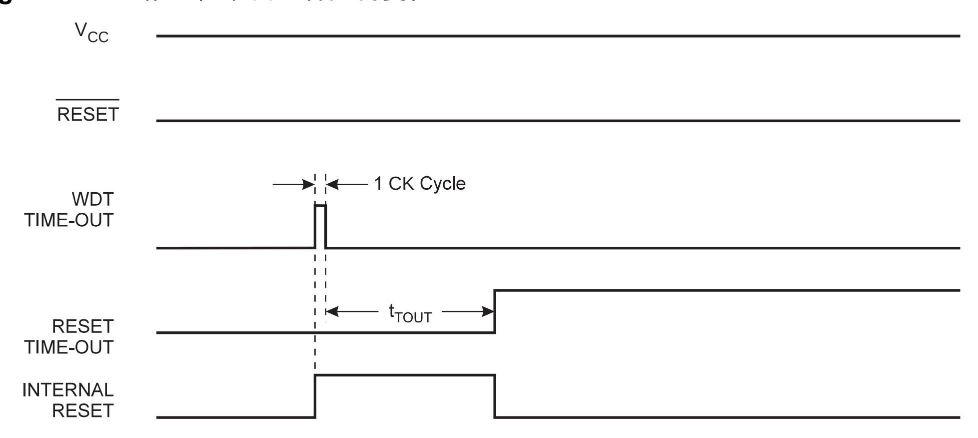




看门狗复位

看门狗定时器溢出时将产生持续时间为 1 个 CK 周期的复位脉冲。在脉冲的下降沿，延时定时器开始对 t_{TOUT} 记数。详见看门狗定时器的具体操作过程。

Figure 27. 工作过程中发生看门狗复位



MCU 控制和状态寄存器 - MCUCSR

MCU 控制和状态寄存器提供了有关引起 MCU 复位的复位源的信息。

Bit	7	6	5	4	3	2	1	0	
	JTD	–	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
读 / 写	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	参见各个位的说明					

在 ATmega103 兼容模式下只有 EXTRF 和 PORF 存在。

- **Bit 4 – JTRF: JTAG 复位标志**

通过 JTAG 指令 AVR_RESET 可以使 JTAG 复位寄存器置位，并引发 MCU 复位，并使 JTRF 置位。上电复位将使其清零，也可以通过写 ”0” 来清除。

- **Bit 3 – WDRF: 看门狗复位标志**

看门狗复位发生时置位。上电复位将使其清零，也可以通过写 ”0” 来清除。

- **Bit 2 – BORF: 掉电检测复位标志**

掉电检测复位发生时置位。上电复位将使其清零，也可以通过写 ”0” 来清除。

- **Bit 1 – EXTRF: 外部复位标志**

外部复位发生时置位。上电复位将使其清零，也可以通过写 ”0” 来清除。

- **Bit 0 – PORF: 上电复位标志**

上电复位发生时置位。只能通过写 ”0” 来清除。

为了使用这些复位标志来识别复位条件，用户应该尽早读取此寄存器的数据，然后将其复位。如果在其他复位发生之前将此寄存器复位，则后续复位源可以通过检查复位标志来了解。

片内基准电压

ATmega128 具有片内能隙基准源，用于掉电检测，或者是作为模拟比较器或 ADC 的输入。 ADC 的 2.56V 基准电压由此片内能隙基准源产生。

基准电压使能信号和启动时间

电压基准的启动时间可能影响其工作方式。启动时间列于 Table 20。为了降低功耗，可以控制基准源仅在如下情况打开：

1. BOD 使能 (熔丝位 BODEN 被编程)。
2. 能隙基准源连接到模拟比较器 (ACSR 寄存器的 ACBG 置位)。
3. ADC 使能。

因此，当 BOD 被禁止时，置位 ACBG 或使能 ADC 后要启动基准源。为了降低掉电模式的功耗，用户可以禁止上述三种条件，并在进入掉电模式之前关闭基准源。

Table 20. 内部电压基准源的特性

符号	参数	最小值	典型值	最大值	单位
V_{BG}	能隙基准源电压	1.15	1.23	1.40	V
t_{BG}	能隙基准源启动时间		40	70	µs
I_{BG}	能隙基准源功耗		10		µA

看门狗定时器

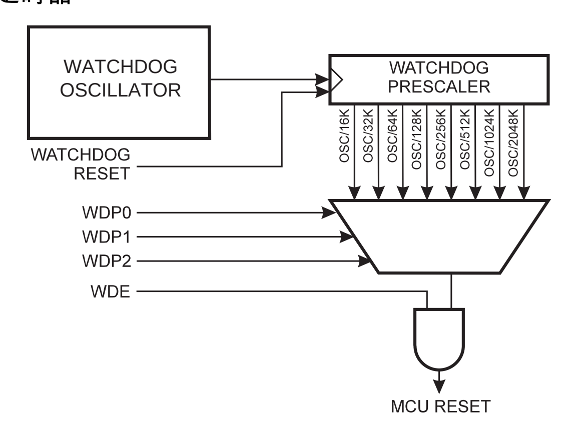
看门狗定时器由独立的 1 Mhz 片内振荡器驱动。这是 V_{CC} = 5V 时的典型值。请参见特性数据以了解其他 V_{CC} 电平下的典型值。通过设置看门狗定时器的预分频器可以调节看门狗复位的时间间隔，如 P 53Table 22 所示。看门狗复位指令 WDR 用来复位看门狗定时器。此外，禁止看门狗定时器或发生复位时定时器也被复位。复位时间有 8 个选项。如果没有及时复位定时器，一旦时间超过复位周期，ATmega128 就复位，并执行复位向量指向的程序。具体的看门狗复位时序在 P 50 有说明。

为了防止无意之间禁止看门狗定时器或改变了复位时间，根据熔丝位 M103C 和 WDTON 芯片提供了 3 个不同的保护级别，如 Table 21. 所示。安全级别 0 相应于 ATmega103 的设置。使能看门狗定时器则没有限制。请参考 P 54“ 改变看门狗定时器配置的时间序列 ” 。

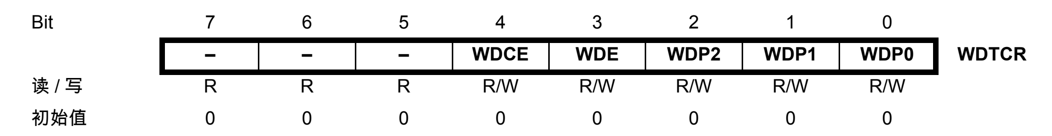
Table 21. WDT 配置表

M103C	WDTON	安全级别	WDT 初始状态	如何禁止 WDT	如何改变复位间隔时间
未编程	未编程	1	禁止	时间序列	时间序列
未编程	已编程	2	使能	总是使能	时间序列
已编程	未编程	0	禁止	时间序列	没有限制
已编程	已编程	2	使能	总是使能	时间序列

Figure 28. 看门狗定时器



看门狗定时器控制寄存器 - WDTCR



Bit	7	6	5	4	3	2	1	0	
	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0	WDTCR
读 / 写	R	R	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bits 7..5 – Res: 保留**

保留位，读操作返回值为零。

- **Bit 4 – WDCE: 看门狗修改使能**

清零 WDE 时必须先置位 WDCE，否则不能禁止看门狗。一旦置位，硬件将在紧接的 4 个时钟周期之后将其清零。请参考有关 WDE 的说明来禁止看门狗。工作于安全级别 1 和 2 时也必须置位 WDCE 以修改预分频器的数据，如 P 54 “ 改变看门狗定时器配置的时间序列 ” 所示。

- **Bit 3 – WDE: 看门狗使能**

WDE为"1“时，看门狗使能，否则看门狗将被禁止。只有在WDCE为"1“时WDE才能清零。以下为关闭看门狗的步骤：

1. 在同一个指令内对 WDCE 和 WDE 写 "1“，即使 WDE 已经为 "1“。
2. 在紧接的 4 个时钟周期之内对 WDE 写 "0”。

工作于安全级别 2 时是永远无法禁止看门狗定时器的。参见 P 54 “ 改变看门狗定时器配置的时间序列 ”。

- **Bits 2..0 – WDP2, WDP1, WDP0: 看门狗定时器预分频器 2, 1, 和 0**

WDP2、 WDP1 和 WDP0 决定看门狗定时器的预分频器，如 Table 22 所示。

Table 22. 看门狗定时器预分频器选项

WDP2	WDP1	WDP0	WDT 振荡器周期	V_{CC} = 3.0V 时典型的溢出周期	V_{CC} = 5.0V 时典型的溢出周期
0	0	0	16K (16,384)	14.8 ms	14.0 ms
0	0	1	32K (32,768)	29.6 ms	28.1 ms
0	1	0	64K (65,536)	59.1 ms	56.2 ms
0	1	1	128K (131,072)	0.12 s	0.11 s
1	0	0	256K (262,144)	0.24 s	0.22 s
1	0	1	512K (524,288)	0.47 s	0.45 s
1	1	0	1,024K (1,048,576)	0.95 s	0.9 s
1	1	1	2,048K (2,097,152)	1.9 s	1.8 s

下面的例子分别用汇编和 C 实现了关闭 WDT 的操作。在此假定中断处于用户控制之下（比如禁止全局中断），因而在执行下面程序时中断不会发生。

汇编代码例程

```
WDT_off:
  ; 置位 WDCE 和 WDE
  ldi  r16, (1<<WDCE)|(1<<WDE)
  out  WDTCR, r16
  ; 关闭 WDT
  ldi  r16, (0<<WDE)
  out  WDTCR, r16
  ret
```

C 代码例程

```
void WDT_off(void)
{
  /* 置位 WDCE 和 WDE */
  WDTCR = (1<<WDCE) | (1<<WDE);
  /* 关闭 WDT */
  WDTCR = 0x00;
}
```

改变看门狗定时器配置的时间序列

改变配置的序列根据不同的安全级别略有不同。

安全级别 0

这个模式与 ATmega103 的看门狗操作相兼容。看门狗的初始状态是禁止的，可以没有限制地通过置位 WDE 来使能它，以及改变定时器溢出周期。禁止看门狗定时器时则需要遵守有关 WDE 的说明。

安全级别 1

在这个模式下，看门狗定时器的初始状态是禁止的，可以没有限制地通过置位 WDE 来使能它。改变定时器溢出周期及禁止（已经使能的）看门狗定时器时需要执行一个特定的时间序列：

1. 在同一个指令内对 WDCE 和 WDE 写 "1“，即使 WDE 已经为 "1“。
2. 在紧接的 4 个时钟周期之内同时对 WDE 写 "0”，以及为 WDP 写入合适的数据，而 WDCE 则写 "0”。

安全级别 2

在这个模式下，看门狗定时器总是使能的， WDE 的读返回值为 "1”。改变定时器溢出周期需要执行一个特定的时间序列：

1. 在同一个指令内对WDCE和WDE写"1“。虽然WDE总是为置位状态，也必须写"1“以启动时序。
2. 在紧接的 4 个时钟周期之内同时对 WDCE 写 "0”，以及为 WDP 写入合适的数据。WDE 的数值可以任意。

中断

本节说明 ATmega128 的中断处理。更一般的 AVR 中断处理请参见 P 12“ 复位和中断处理 ”。

ATmega128 的中断向量

Table 23. 复位和中断向量

向量号	程序地址 [2]	中断源	中断定义
1	$0000[1]	RESET	外部引脚，上电复位，掉电检测复位，看门狗复位，以及 JTAG AVR 复位
2	$0002	INT0	外部中断请求 0
3	$0004	INT1	外部中断请求 1
4	$0006	INT2	外部中断请求 2
5	$0008	INT3	外部中断请求 3
6	$000A	INT4	外部中断请求 4
7	$000C	INT5	外部中断请求 5
8	$000E	INT6	外部中断请求 6
9	$0010	INT7	外部中断请求 7
10	$0012	TIMER2 COMP	定时器 / 计数器 2 比较匹配
11	$0014	TIMER2 OVF	定时器 / 计数器 2 溢出
12	$0016	TIMER1 CAPT	定时器 / 计数器 1 捕捉事件
13	$0018	TIMER1 COMPA	定时器 / 计数器 1 比较匹配 A
14	$001A	TIMER1 COMPB	定时器 / 计数器 1 比较匹配 B
15	$001C	TIMER1 OVF	定时器 / 计数器 1 溢出
16	$001E	TIMER0 COMP	定时器 / 计数器 0 比较匹配
17	$0020	TIMER0 OVF	定时器 / 计数器 0 溢出
18	$0022	SPI, STC	SPI 串行传输结束
19	$0024	USART0, RX	USART0, Rx 结束
20	$0026	USART0, UDRE	USART0 数据寄存器空
21	$0028	USART0, TX	USART0, Tx 结束
22	$002A	ADC	ADC 转换结束
23	$002C	EE READY	EEPROM 就绪
24	$002E	ANALOG COMP	模拟比较器
25	$0030[3]	TIMER1 COMPC	定时器 / 计数器 1 比较匹配 C
26	$0032[3]	TIMER3 CAPT	定时器 / 计数器 3 捕捉事件
27	$0034[3]	TIMER3 COMPA	定时器 / 计数器 3 比较匹配 A
28	$0036[3]	TIMER3 COMPB	定时器 / 计数器 3 比较匹配 B
29	$0038[3]	TIMER3 COMPC	定时器 / 计数器 3 比较匹配 C
30	$003A[3]	TIMER3 OVF	定时器 / 计数器 3 溢出
31	$003C[3]	USART1, RX	USART1, Rx 结束



Table 23. 复位和中断向量

向量号	程序 地址 (2)	中断源	中断定义
32	$003E(3)	USART1, UDRE	USART1 数据寄存器空
33	$0040(3)	USART1, TX	USART1, Tx 结束
34	$0042(3)	TWI	两线串行接口
35	$0044(3)	SPM READY	保存程序存储器内容就绪

Notes: 1. 当熔丝位BOOTRST被编程时，复位后程序跳转到Boot Loader。请参见 P 255"支持引导装入程序 – 在写的同时可以读 (RWW, Read-While-Write) 的自我编程能力 " 。
2. 当寄存器MCUCR的IVSEL置位时，中断向量转移到Boot区的起始地址。此时各个中断向量的实际地址为表中地址与 Boot 区起始地址之和。
3. 地址为 $0030 - $0044 的中断在 ATmega103 兼容模式中不存在。

Table 24给出了不同的BOOTRST/IVSEL设置下的复位和中断向量的位置。如果程序永远不使能中断，中断向量就没有意义。用户可以在此直接写程序。同样，如果复位向量位于应用区，而其他中断向量位于 Boot 区，则复位向量之后可以直接写程序。反过来亦是如此。

Table 24. 复位和中断向量位置的确定

BOOTRST	IVSEL	复位地址	中断向量起始地址
1	0	$0000	$0002
1	1	$0000	Boot 区复位地址 + $0002
0	0	Boot 区复位地址	$0002
0	1	Boot 区复位地址	Boot 区复位地址 + $0002

Note: Boot 区复位地址列 P 266Table 112 。对于熔丝位 BOOTRST ,"1" 表示未编程，"0" 表示已编程。

ATmega128 典型的复位和中断设置如下：

```
地址        符号     代码            说明
$0000       jmp      RESET       ; 复位句柄
$0002       jmp      EXT_INT0    ; IRQ0 句柄
$0004       jmp      EXT_INT1    ; IRQ1 句柄
$0006       jmp      EXT_INT2    ; IRQ2 句柄
$0008       jmp      EXT_INT3    ; IRQ3 句柄
$000A       jmp      EXT_INT4    ; IRQ4 句柄
$000C       jmp      EXT_INT5    ; IRQ5 句柄
$000E       jmp      EXT_INT6    ; IRQ6 句柄
$0010       jmp      EXT_INT7    ; IRQ7 句柄
$0012       jmp      TIM2_COMP   ; 定时器 2 比较句柄
$0014       jmp      TIM2_OVF    ; 定时器 2 溢出句柄
$0016       jmp      TIM1_CAPT   ; 定时器 1 捕捉句柄
$0018       jmp      TIM1_COMPA  ; 定时器 1 比较 A 句柄
$001A       jmp      TIM1_COMPB  ; 定时器 1 比较 B 句柄
$001C       jmp      TIM1_OVF    ; 定时器 1 溢出句柄
$001E       jmp      TIM0_COMP   ; 定时器 0 比较句柄
$0020       jmp      TIM0_OVF    ; 定时器 0 溢出句柄
$0022       jmp      SPI_STC     ; SPI 传输结束句柄
$0024       jmp      USART0_RXC  ; USART0 接收结束句柄
$0026       jmp      USART0_DRE  ; USART0,UDR 空句柄
$0028       jmp      USART0_TXC  ; USART0 发送结束句柄
$002A       jmp      ADC         ; ADC 转换结束句柄
$002C       jmp      EE_RDY      ; EEPROM 就绪句柄
$002E       jmp      ANA_COMP    ; 模拟比较器句柄
$0030       jmp      TIM1_COMPC  ; 定时器 1 比较 C 句柄
$0032       jmp      TIM3_CAPT   ; 定时器 3 捕捉句柄
$0034       jmp      TIM3_COMPA  ; 定时器 3 比较 A 句柄
$0036       jmp      TIM3_COMPB  ; 定时器 3 比较 B 句柄
$0038       jmp      TIM3_COMPC  ; 定时器 3 比较 C 句柄
$003A       jmp      TIM3_OVF    ; 定时器 3 溢出句柄
$003C       jmp      USART1_RXC  ; USART1 接收结束句柄
$003E       jmp      USART1_DRE; USART1,UDR 空句柄
$0040       jmp      USART1_TXC  ; USART1 发送结束句柄
$0042       jmp      TWI         ; 两线串行接口中断句柄
$0044       jmp      SPM_RDY     ; SPM 就绪句柄
;
$0046    RESET:ldir16, high(RAMEND); 主程序
$0047       out      SPH,r16     ; 设置堆栈指针为 RAM 的顶部
$0048       ldi      r16, low(RAMEND)
$0049       out      SPL,r16
$004A       sei                  ; 使能中断
$004B       <instr>  xxx
  ...       ...      ...    ...
```

当熔丝位 BOOTRST 未编程，Boot 区为 8K 字节，且中断使能之前寄存器 MCUCR 的 IVSEL 置位时，典型的复位和中断设置如下：

```
地址        符号     代码            说明
$0000    RESET:ldi   r16,high(RAMEND) ; 主程序
```



```
$0001         out     SPH,r16     ; 设置堆栈指针为 RAM 的顶部
$0002         ldi     r16,low(RAMEND)
$0003         out     SPL,r16
$0004         sei                 ; 使能中断
$0005         <instr>  xxx
;
.org $F002
$F002         jmp     EXT_INT0    ; IRQ0 句柄
$F004         jmp     EXT_INT1    ; IRQ1 句柄
...           ...     ...         ;
$F044         jmp     SPM_RDY     ; SPM 就绪句柄
```

当熔丝位 BOOTRST 已编程，且 Boot 区为 8K 字节时，典型的复位和中断设置如下：

```
地址          符号    代码                说明
.org $0002
$0002         jmp     EXT_INT0    ; IRQ0 句柄
$0004           jmp     EXT_INT1    ; IRQ1 句柄
...             ...     ...         ;
$0044           jmp     SPM_RDY     ; SPM 就绪句柄
;
.org $F000
$F000    RESET:  ldi     r16,high(RAMEND) ; 主程序
$F001           out     SPH,r16     ; 设置堆栈指针为 RAM 的顶部
$F002           ldi     r16,low(RAMEND)
$F003           out     SPL,r16
$F004           sei                 ; 使能中断
$F005           <instr>  xxx
```

当熔丝位 BOOTRST 已编程，Boot 区为 8K 字节时，且中断使能之前寄存器 MCUCR 的 IVSEL 置位时，典型的复位和中断设置如下：

```
地址          符号    代码                说明
;
.org $F000
$F000           jmp     RESET       ; Reset 句柄
$F002           jmp     EXT_INT0    ; IRQ0 句柄
$F004           jmp     EXT_INT1    ; IRQ1 句柄
...             ...     ...         ;
$F044           jmp     SPM_RDY   ; SPM 就绪句柄
$F046    RESET:  ldi     r16,high(RAMEND) ; 主程序
$F047           out     SPH,r16     ; 设置堆栈指针为 RAM 的顶部
$F048           ldi     r16,low(RAMEND)
$F049           out     SPL,r16
$F04A           sei                 ; 使能中断
$F04B           <instr>  xxx
```

在应用区和 Boot 区之间移动中断

通用中断控制寄存器决定中断向量表的放置地址。

MCU 控制寄存器 - MCUCR

Bit	7	6	5	4	3	2	1	0	
	SRE	SRW10	SE	SM1	SM0	SM2	IVSEL	IVCE	MCUCR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

初始值	0	0	0	0	0	0	0	0

- **Bit 1 – IVSEL: 中断向量选择**

当 IVSEL 为 "0“ 时，中断向量位于 Flash 存储器的起始地址；当 IVSEL 为 "1“ 时，中断向量转移到 Boot 区的起始地址。实际的 Boot 区起始地址由熔丝位 BOOTSZ 确定。具体请参考 P 255“ 支持引导装入程序 – 在写的同时可以读 (RWW, Read-While-Write) 的自我编程能力 ” 。为了防止无意之间改变中断向量表，修改 IVSEL 时需要遵照如下过程：

1. 置位中断向量修改使能位 IVCE。

在紧接的 4 个时钟周期里将需要的数据写入 IVSEL，同时对 IVCE 写 ”0”。

执行上述序列时中断自动被禁止。其实，在置位 IVCE 时中断就被禁止了，并一直保持到写 IVSEL 操作之后的下一条语句。如果没有 IVSEL 写操作，则中断在置位 IVCE 之后的 4 个时钟周期保持禁止。状态寄存器的位 I 不受此序列的影响。

Note: 若中断向量位于Boot区，且Boot锁定位BLB02被编程，则执行应用区的程序时中断被禁止；若中断向量位于应用区，且 Boot 锁定位 BLB12 被编程，则执行 Boot 区的程序时中断被禁止。有关 Boot 锁定位的细节请参见 P 255“ 支持引导装入程序 – 在写的同时可以读 (RWW, Read-While-Write) 的自我编程能力 ” 。



- **Bit 0 – IVCE: 中断向量修改使能**

改变 IVSEL 时 IVCE 必须置位。在 IVCE 或 IVSEL 写操作之后 4 个时钟周期， IVCE 被硬件清零。如前面所述，置位 IVCE 将禁止中断。代码如下：

汇编代码例程

```
Move_interrupts:
  ; 使能中断向量的修改
  ldi  r16, (1<<IVCE)
  out  MCUCR, r16
  ; 将中断向量转移到 boot 区
  ldi  r16, (1<<IVSEL)
  out  MCUCR, r16
  ret
```

C 代码例程

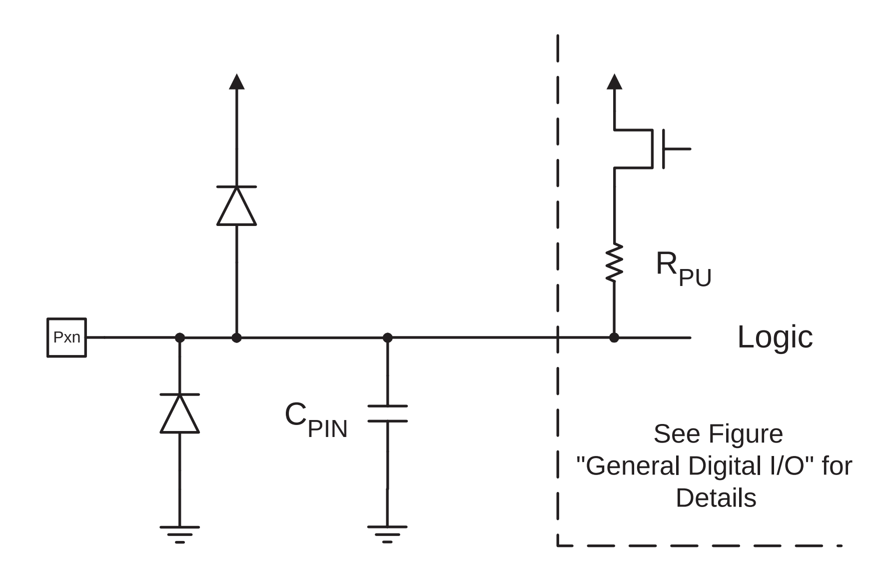
```
void Move_interrupts(void)
{
  /* 使能中断向量的修改 */
  MCUCR = (1<<IVCE);
  /* 将中断向量转移到 boot 区 */
  MCUCR = (1<<IVSEL);
}
```

I/O 端口

介绍

作为通用数字 I/O 使用时，所有 AVR I/O 端口都具有真正的读 - 修改 - 写功能。这意味着用 SBI 或 CBI 指令改变某些管脚的方向 (或者是端口电平、禁止 / 使能上拉电阻) 时不会无意地改变其他管脚的方向 (或者是端口电平、禁止 / 使能上拉电阻)。输出缓冲器具有对称的驱动能力，可以输出或吸收大电流，直接驱动 LED。所有的端口引脚都具有与电压无关的上拉电阻。并有保护二极管与 V_{CC} 和地相连，如 Figure 29 所示。请参见 P 299“ 电气特性 ” 以获取完整的参数列表。

Figure 29. I/O 引脚等效原理图



本节所有的寄存器和位以通用格式表示：小写的 “x” 表示端口的序号，而小写的 “n” 代表位的序号。但是在程序里要写完整。例如， PORTB3 表示端口 B 的第 3 位，而本节的通用格式为 PORTxn。物理 I/O 寄存器和位定义列于 P 81“I/O 端口寄存器的说明 ” 。

每个端口都有三个 I/O 存储器地址：数据寄存器 – PORTx、数据方向寄存器 – DDRx 和端口输入引脚 – PINx。数据寄存器和数据方向寄存器为读 / 写寄存器，而端口输入引脚为只读寄存器。当寄存器 SFIOR 的上拉禁止位 PUD 置位时所有端口的全部引脚的上拉电阻都被禁止。

作为通用数字 I/O 时的端口请参见 P 62“ 作为通用数字 I/O 的端口 ” 。多数端口引脚是与第二功能复用的，如 P 66“ 端口的第二功能 ” 所示。请参见各个模块的具体说明以了解引脚的第二功能。

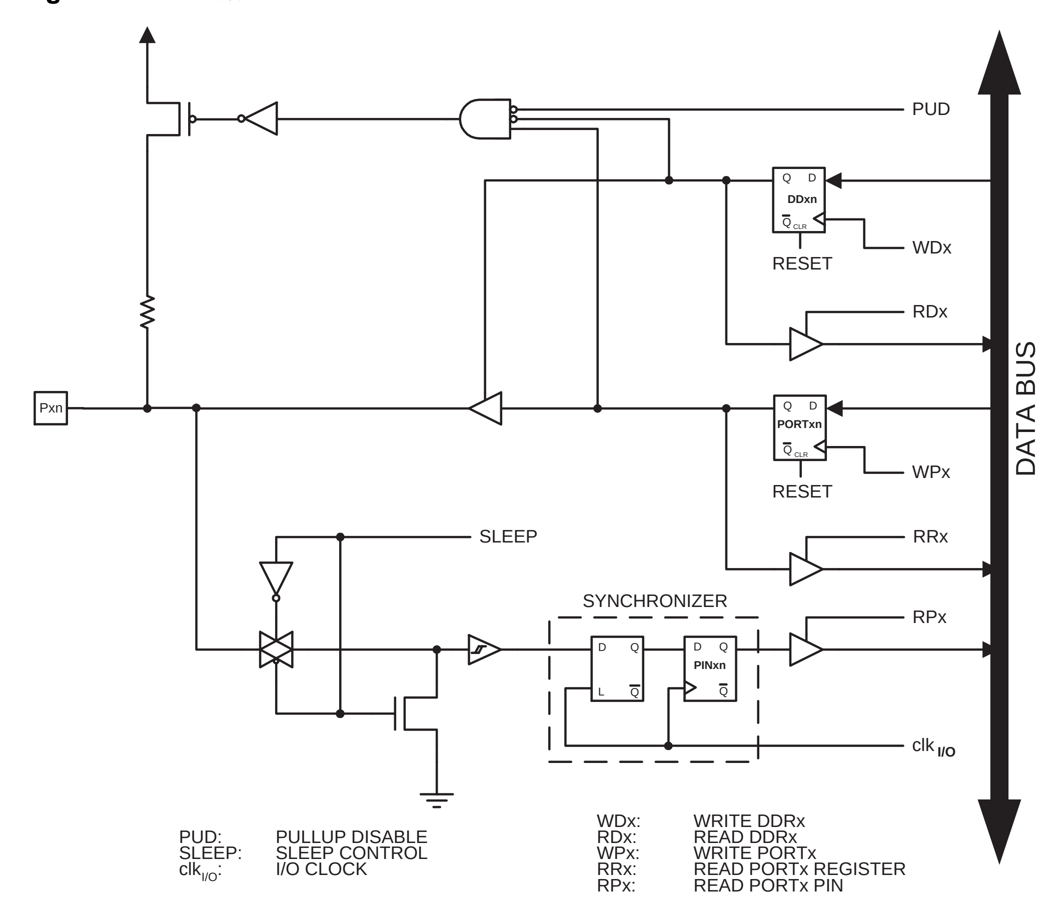
使能某些引脚的第二功能不会影响其他引脚用于通用数字 I/O。



作为通用数字 I/O 的端口

端口为具有上拉电阻 (可选的功能) 的双向 I/O。 Figure 30 为一个 I/O 端口引脚的说明。

Figure 30. 通用数字 I/O(1)



Note: 1. WPx, WDx, RRx, RPx 和 RDx 对于同一端口的所有引脚都是一样的。$clk_{I/O}$, SLEEP 和 PUD 则对所有的端口都是一样的。

配置引脚

每个端口引脚都具有三个寄存器位：DDxn、PORTxn 和 PINxn，如 P 81“I/O 端口寄存器的说明 ” 所示。 DDxn 位于 DDRx 寄存器， PORTxn 位于 PORTx 寄存器， PINxn 位于 PINx 寄存器。

DDxn 以来选择引脚的方向。当 DDxn 为 "1“ 时， Pxn 配置为输出；否则为输入。

当引脚配置为输入时，若 PORTxn 为 "1“，上拉电阻将使能。如果需要关闭这个上拉电阻，可以将 PORTxn 清零，或者将这个引脚配置为输出。复位时各引脚为三态，即使此时没有时钟在运行。

当引脚配置为输出时，若 PORTxn 为 "1“，引脚输出高电平 ("1“)，否则输出低电平 (“0“)。

在 (高阻态) 三态 ({DDxn, PORTxn} = 0b00) 输出高电平 ({DDxn, PORTxn} = 0b11) 两种状态之间进行切换时，上拉电阻使能 ({DDxn, PORTxn} = 0b01) 或输出低电平 ({DDxn, PORTxn} = 0b10) 这两种模式必然会有一个发生。通常，上拉电阻使能是完全可以接受的，因为高阻环境不在意是强高电平输出还是上拉输出。如果使用情况不是这样子，可以通过置位 SFIOR 寄存器的 PUD 来禁止所有端口的上拉电阻。

在上拉输入和输出低电平之间切换也有同样的问题。用户必须选择高阻态 ({DDxn, PORTxn} = 0b00) 或输出高电平 ({DDxn, PORTxn} = 0b11) 作为中间步骤。

Table 25 总结了引脚的控制信号。

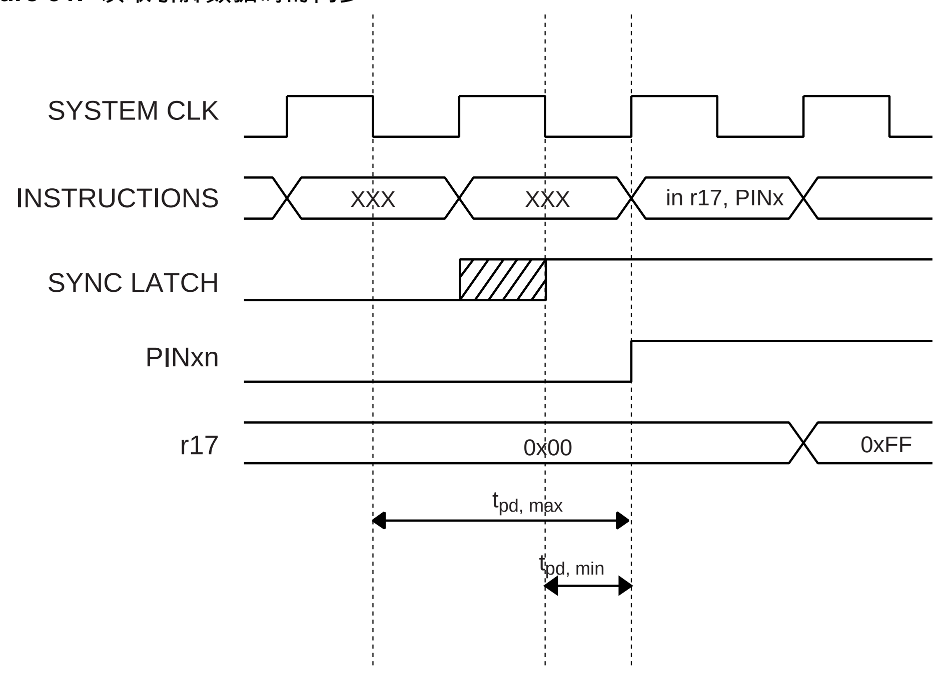
Table 25. 端口引脚配置

DDxn	PORTxn	PUD (in SFIOR)	I/O	上拉电阻	说明
0	0	X	输入	No	高阻态 (Hi-Z)
0	1	0	输入	Yes	被外部电路拉低时将输出电流
0	1	1	输入	No	高阻态 (Hi-Z)
1	0	X	输出	No	输出低电平 (吸收电流)
1	1	X	输出	No	输出高电平 (输出电流)

读取引脚上的数据

不论如何配置 DDxn , 都可以通过读取 PINxn 寄存器来获得引脚电平。如 Figure 30 所示 , PINxn 寄存器的各个位与其前面的锁存器组成了一个同步器。这样就可以避免在内部时钟状态发生改变的短时间范围内由于引脚电平变化而造成的信号不稳定。其缺点是引入了延迟。 Figure 31 为读取引脚电平时同步器的时序图。最大和最小传输延迟分别为 $t_{pd,max}$ 和 $t_{pd,min}$。

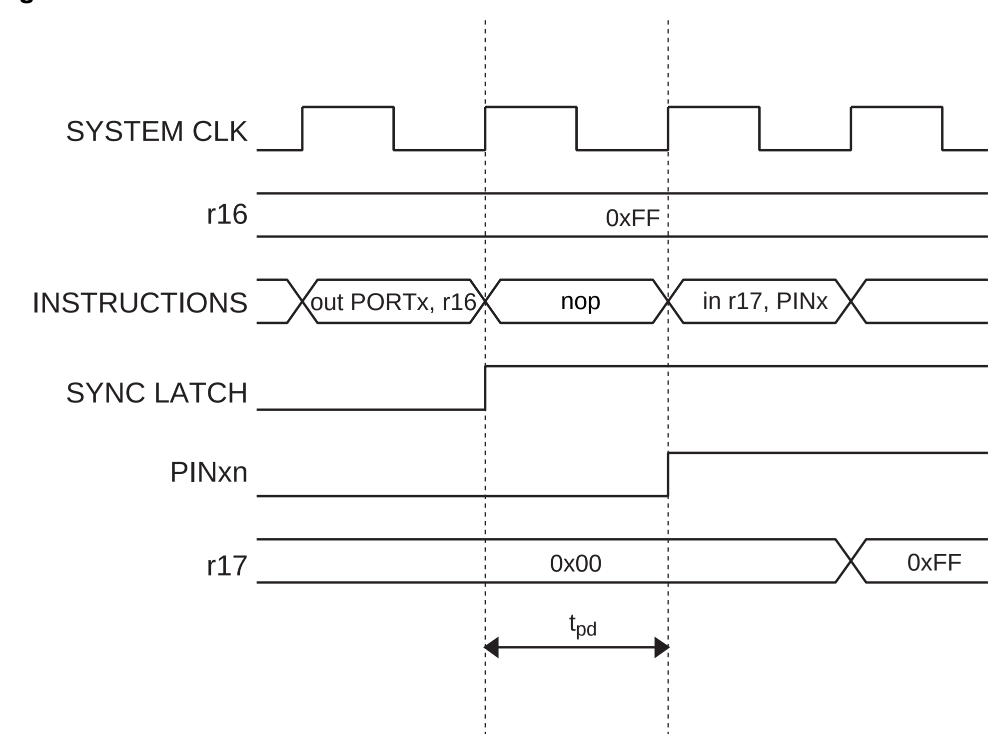
Figure 31. 读取引脚数据时的同步



下面考虑第一个系统时钟下降沿之后起始的时钟周期。当时钟信号为低时锁存器是关闭的 ; 而时钟信号为高时信号可以自由通过 , 如图中 SYNC LATCH 信号的阴影区所示。时钟为低时信号即被锁存 , 然后在紧接着的系统时钟上升沿锁存到 PINxn 寄存器。如 $t_{pd,max}$ 和 $t_{pd,min}$ 所示 , 引脚上的信号转换延迟界于 ½ ~ 1½ 个系统时钟。

如 Figure 32 所示 , 读取软件赋予的引脚电平时需要在赋值指令 *out* 和读取指令 *in* 之间有一个时钟周期的间隔 , 如 *nop* 指令。*out* 指令在时钟的上升沿置位 SYNC LATCH 信号。此时同步器的延迟时间 t_{pd} 为一个系统时钟。

Figure 32. 读取软件赋予的引脚电平的同步



下面的例子演示了如何置位端口 B 的引脚 0 和 1，清零引脚 2 和 3，以及将引脚 4 到 7 设置为输入，并且为引脚 6 和 7 设置上拉电阻。然后将各个引脚的数据读回来。如前面讨论的那样，我们在输出和输入语句之间插入了一个 *nop* 指令。

汇编代码例程 (1)

```
...
; 定义上拉电阻和设置高电平输出
; 为端口引脚定义方向
ldi   r16,(1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)
ldi   r17,(1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0)
out   PORTB,r16
out   DDRB,r17
; 为了同步插入 nop 指令
nop
; 读取端口引脚
in    r16,PINB
...
```

C 代码例程 (1)

```
unsigned char i;
  ...
  /* 定义上拉电阻和设置高电平输出 */
  /* 为端口引脚定义方向 */
  PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
  DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
  /* 为了同步插入 nop 指令 */
  _NOP();
  /* 读取端口引脚 */
  i = PINB;
  ...
```

Note: 1. 在汇编程序里使用了两个暂存器。其目的是为了使整个操作过程的时间最短。

数字输入使能和睡眠模式

如 Figure 30 所示，数字输入信号 (施密特触发器的输入) 可以钳位到地。图中的 SLEEP 信号由 MCU 的睡眠控制器在各种睡眠模式下设置，以防止在输入悬空或模拟输入电平接近 $V_{CC}/2$ 时消耗太多的电流。

引脚作为外部中断输入时 SLEEP 信号无效。在使能引脚的第二功能时 SLEEP 也让位于第二功能，如 P 66" 端口的第二功能 " 里描述的那样。

当异步外部中断引脚配置为任意逻辑电平变化都可以引发中断时，如果外部中断没有使能，而引脚上又施加了高电平，则 MCU 从睡眠模式唤醒时相应的外部中断标志将置位。因为在睡眠时由于 SLEEP 信号的作用内部信号被钳位到地，而唤醒后外部高电平输入到内部逻辑，产生了低电平到高电平的信号变化。

未连接引脚的处理

如果有引脚未被使用，建议给这些引脚赋予一个确定电平。虽然如上文所述，在深层休眠模式下大多数数字输入被禁用，但还是需要避免因引脚没有确定的电平而造成悬空引脚在其它数字输入使能模式 (复位、工作模式、空闲模式) 消耗电流。

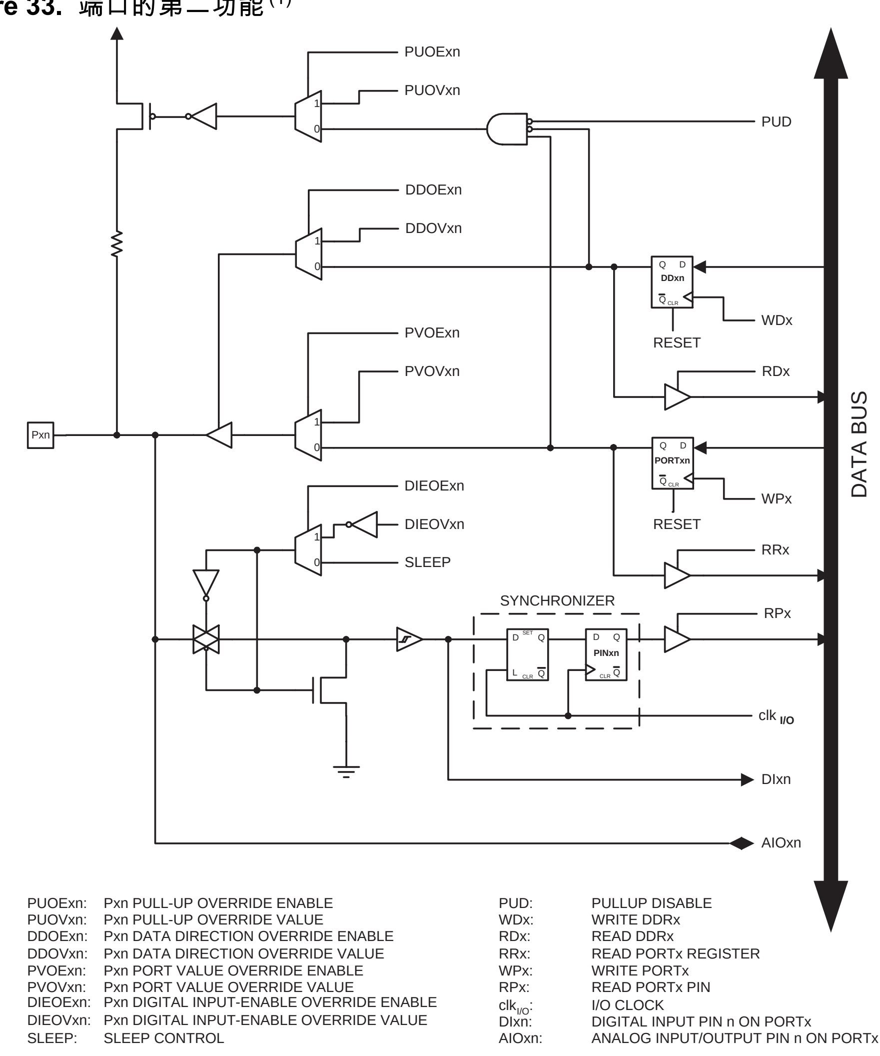
最简单的保证未用引脚具有确定电平的方法是使能内部上拉电阻。但要注意的是复位时上拉电阻将被禁用。如果复位时的功耗也有严格要求则建议使用外部上拉或下拉电阻。不推荐直接将未用引脚与 V_{CC} 或 GND 连接，因为这样可能会在引脚偶然作为输出时出现冲击电流。



端口的第二功能

除了一般的数字 I/O 之外，大多数端口引脚都具有第二功能。 Figure 33 说明了由 Figure 30 简化出来的端口引脚控制信号是如何被第二功能所重载的。这些被重载的信号不会出现在所有的端口引脚，但本图可以看作是适用于 AVR 系列处理器所有端口引脚的一般说明。

Figure 33. 端口的第二功能 (1)



Note: 1. WPx, WDx, RLx, RPx和RDx对于同一个端口的所有引脚都是一样的。$clk_{I/O}$, SLEEP和 PUD 则对所有的端口都是一样的。其他信号则只对某一个信号有效

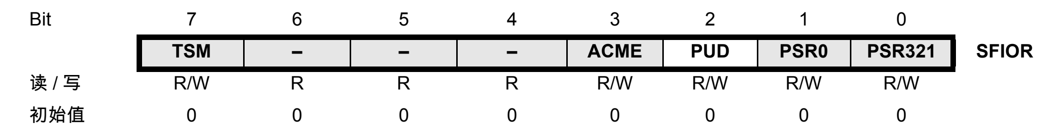
Table 26 为被第二功能重载的信号的简介。表中没有给出 Figure 33 的引脚和端口索引。

Table 26. 第二功能的说明

信号名称	全 称	说 明
PUOE	上拉电阻 重载使能	若此信号置位，上拉电阻使能将受控于 PUOV ；若此信号清零，则 {DDxn, PORTxn, PUD} = 0b010 时上拉电阻使能
PUOV	上拉电阻 重载使能	若 PUOE 置位，则 PUOV 置位 / 清零时上拉电阻使能 / 禁止，而不管 DDxn、PORTxn 和 PUD 寄存器各个位的设置如何
DDOE	数据方向 重载使能	如果此信号置位，则输出驱动使能由 DDOV 控制；若此信号清零，输出驱动使能由 DDxn 寄存器控制
DDOV	数据方向 重载使能	若 DDOE 置位，则 DDOV 置位 / 清零时输出驱动使能 / 禁止，而不管 DDxn 寄存器的设置如何
PVOE	端口数据 重载使能	如果这个信号置位，且输出驱动使能，端口数据由 PVOV 控制；若 PVOE 清零，且输出驱动使能，端口数据由寄存器 PORTxn 控制
PVOV	端口数据 重载使能	如 PVOE 设置，端口值设置为 PVOV，而不管寄存器 PORTxn 如何设置
DIEOE	数字输入 重载使能	如果这个信号置位，数字输入使能由 DIEOV 控制；若 DIEOE 清零，数字输入使能由 MCU 的状态确定 (正常模式，睡眠模式)
DIEOV	数字输入 重载使能	若 DIEOE 置位， DIEOV 置位 / 清零时数字输入使能 / 禁止，而不管 MCU 的状态如何 (正常模式，睡眠模式)
DI	数字输入	此信号为第二功能的数字输入。在图中，这个信号与施密特触发相连，并且在同步器之前。除非数字输入用作时钟源，否则第二功能模块将使用自己的同步器
AIO	模拟信号 输入 / 输出	模拟输入 / 输出。信号直接与引脚接点相连，而且可以用作双向端口

下面的几小节将简单地说明每个端口的第二功能以及相关的信号。具体请参考有关第二功能的说明。

特殊功能 IO 寄存器 - SFIOR



Bit	7	6	5	4	3	2	1	0	
	TSM	–	–	–	ACME	PUD	PSR0	PSR321	SFIOR
读 / 写	R/W	R	R	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 2 – PUD: 禁止上拉电阻**

置位时，即使将寄存器 DDxn 和 PORTxn 配置为使能上拉电阻 ({DDxn, PORTxn} = 0b01)，I/O 端口的上拉电阻也被禁止。请参见 P 62“ 配置引脚 ” 。

端口 A 的第二功能

端口 A 的第二功能为外部存储器接口的低字节地址以及数据。

Table 27. 端口 A 的第二功能

端口引脚	第二功能
PA7	AD7 (外部存储器接口地址及数据位 7)
PA6	AD6 (外部存储器接口地址及数据位 6)
PA5	AD5 (外部存储器接口地址及数据位 5)
PA4	AD4 (外部存储器接口地址及数据位 4)
PA3	AD3 (外部存储器接口地址及数据位 3)
PA2	AD2 (外部存储器接口地址及数据位 2)
PA1	AD1 (外部存储器接口地址及数据位 1)
PA0	AD0 (外部存储器接口地址及数据位 0)

Table 28 和 Table 29 将端口 A 的第二功能与 P 66Figure 33 的重载信号关联在了一起。

Table 28. PA7..PA4 的第二功能

信号名称	PA7/AD7	PA6/AD6	PA5/AD5	PA4/AD4
PUOE	SRE	SRE	SRE	SRE
PUOV	~($\overline{WR}$ | ADA[(1)]) • PORTA7 • $\overline{PUD}$	~($\overline{WR}$ | ADA) • PORTA6 • $\overline{PUD}$	~($\overline{WR}$ | ADA) • PORTA5 • $\overline{PUD}$	~($\overline{WR}$ | ADA) • PORTA4 • $\overline{PUD}$
DDOE	SRE	SRE	SRE	SRE
DDOV	$\overline{WR}$ | ADA	$\overline{WR}$ | ADA	$\overline{WR}$ | ADA	$\overline{WR}$ | ADA
PVOE	SRE	SRE	SRE	SRE
PVOV	A7 • ADA | D7 OUTPUT • $\overline{WR}$	A6 • ADA | D6 OUTPUT • $\overline{WR}$	A5 • ADA | D5 OUTPUT • $\overline{WR}$	A4 • ADA | D4 OUTPUT • $\overline{WR}$
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	D7 输入	D6 输入	D5 输入	D4 输入
AIO	–	–	–	–

Note: 1. ADA 为地址有效 (ADdress Active) 的缩写，代表地址输出的时间。请参见 P 23“ 外部存储器接口 ” 。

Table 29. PA3..PA0 的第二功能

信号名称	PA3/AD3	PA2/AD2	PA1/AD1	PA0/AD0
PUOE	SRE	SRE	SRE	SRE
PUOV	~(WR | ADA) • PORTA3 • PUD	~(WR | ADA) • PORTA2 • PUD	~(WR | ADA) • PORTA1 • PUD	~(WR | ADA) • PORTA0 • PUD
DDOE	SRE	SRE	SRE	SRE
DDOV	WR | ADA	WR | ADA	WR | ADA	WR | ADA
PVOE	SRE	SRE	SRE	SRE
PVOV	A3 • ADA | D3 OUTPUT • WR	A2• ADA | D2 OUTPUT • WR	A1 • ADA | D1 OUTPUT • WR	A0 • ADA | D0 OUTPUT • WR
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	D3 输入	D2 输入	D1 输入	D0 输入
AIO	–	–	–	–

端口 B 的第二功能

端口 B 的第二功能列于 Table 30。

Table 30. 端口 B 的第二功能

引脚	第二功能
PB7	OC2/OC1C(1) (T/C2 的输出比较和 PWM 输出 , 或是 T/C1 的输出比较和 PWM 输出 C)
PB6	OC1B (T/C1 的输出比较和 PWM 输出 B)
PB5	OC1A (T/C1 的输出比较和 PWM 输出 A)
PB4	OC0 (T/C0 的输出比较和 PWM 输出)
PB3	MISO (SPI 总线的主机输入 / 从机输出信号)
PB2	MOSI (SPI 总线的主机输出 / 从机输入信号)
PB1	SCK (SPI 总线的的串行时钟)
PB0	SS (SPI 从机选择引脚)

Note: 1. ATmega103 兼容模式没有 OC1C。

引脚配置如下 :

- **OC2/OC1C, Bit 7**

OC2 , 输出比较匹配模块的输出 : PB7 可以作为 T/C2 输出比较模块的输出。此时引脚必须配置为输出 (DDB7 设置为 “1”)。 OC2 引脚也是 PWM 模式的输出引脚。

OC1C , 输出比较匹配 C 模块的输出 : PB7 可以作为 T/C1 输出比较 C 模块的输出。此时引脚必须配置为输出 (DDB7 设置为 “1”)。 OC1C 也是 PWM 模式的输出引脚。

- **OC1B, Bit 6**

OC1B , 输出比较匹配 B 模块的输出 : PB6 可以作为 T/C1 输出比较 B 模块的输出。此时引脚必须配置为输出 (DDB6 设置为 “1”)。 OC1B 也是 PWM 模式的输出引脚。

- **OC1A, Bit 5**

OC1A , 输出比较匹配 A 模块的输出 : PB5 可以作为 T/C1 输出比较 A 的输出。此时引脚必须配置为输出 (DDB5 设置为 “1”)。 OC1A 也是 PWM 模式的输出引脚。





- **OC0, Bit 4**

OC0，输出比较匹配模块的输出：PB4 可以作为 T/C0 输出比较模块的输出。此时引脚必须配置为输出 (DDB4 设置为 “1”)。 OC0 也是 PWM 模式的输出引脚。

- **MISO – 端口 B, Bit 3**

MISO: SPI 通道的主机数据输入，从机数据输出。当工作于主机模式时，不论 DDB3 设置如何，这个引脚都将设置为输入。当工作于从机模式时，这个引脚的数据方向由 DDB3 控制。设置为输入后，上拉电阻由 PORTB3 控制。

- **MOSI – 端口 B, Bit 2**

MOSI: SPI 通道的主机数据输出，从机数据输入。当工作于从机模式时，不论 DDB2 设置如何，这个引脚都将设置为输入。当工作于主机模式时，这个引脚的数据方向由 DDB2 控制。设置为输入后，上拉电阻由 PORTB2 控制。

- **SCK – 端口 B, Bit 1**

SCK: SPI 通道的主机时钟输出，从机时钟输入。当工作于从机模式时，不论 DDB1 设置如何，这个引脚都将设置为输入。当工作于主机模式时，这个引脚的数据方向由 DDB1 控制。设置为输入后，上拉电阻由 PORTB1 控制。

- **$\overline{\text{SS}}$ – 端口 B, Bit 0**

$\overline{\text{SS}}$: 从机选择输入。当工作于从机模式时，不论 DDB0 设置如何，这个引脚都将设置为输入。当工作于主机模式时，这个引脚的数据方向由 DDB0 控制。设置为输入后，上拉电阻由 PORTB0 控制。

Table 31和Table 32将端口 B的第二功能与P 66Figure 33 的重载信号关联在了一起。SPI MSTR INPUT和SPI SLAVE OUTPUT构成了MISO信号，而MOSI可以分解为SPI MSTR OUTPUT 和 SPI SLAVE INPUT。

Table 31. PB7..PB4 的第二功能

信号名称	PB7/OC2/OC1C	PB6/OC1B	PB5/OC1A	PB4/OC0
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	OC2/OC1C 使能 (1)	OC1B 使能	OC1A 使能	OC0 使能
PVOV	OC2/OC1C(1)	OC1B	OC1A	OC0B
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	–	–	–	–

Note: 1. 参见 P 145“ 输出比较调制器 (OCM1C2)” 。 ATmega103 兼容模式下没有 OC1C。

Table 32. PB3..PB0 的第二功能

信号名称	PB3/MISO	PB2/MOSI	PB1/SCK	PB0/$\overline{SS}$
PUOE	SPE • MSTR	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$
PUOV	PORTB3 • $\overline{PUD}$	PORTB2 • $\overline{PUD}$	PORTB1 • $\overline{PUD}$	PORTB0 • $\overline{PUD}$
DDOE	SPE • MSTR	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$	SPE • $\overline{MSTR}$
DDOV	0	0	0	0
PVOE	SPE • $\overline{MSTR}$	SPE • MSTR	SPE • MSTR	0
PVOV	SPI 从机输出	SPI 主机输出	SCK 输出	0
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	SPI 主机输入	SPI 从机输入	SCK 输入	SPI $\overline{SS}$
AIO	–	–	–	–

端口 C 的第二功能

在 ATmega103 兼容模式下 , 端口 C 为输出端口。其第二功能为外部存储器接口的地址高字节。

Table 33. 端口 C 的第二功能

端口引脚	第二功能
PC7	A15
PC6	A14
PC5	A13
PC4	A12
PC3	A11
PC2	A10
PC1	A9
PC0	A8

Table 34 和 Table 35 将端口 C 的第二功能与 P 66Figure 33 的重载信号关联在了一起。

Table 34. PC7..PC4 的第二功能

信号名称	PC7/A15	PC6/A14	PC5/A13	PC4/A12
PUOE	SRE • (XMM[1]<1)	SRE • (XMM<2)	SRE • (XMM<3)	SRE • (XMM<4)
PUOV	0	0	0	0
DDOE	SRE • (XMM<1)	SRE • (XMM<2)	SRE • (XMM<3)	SRE • (XMM<4)
DDOV	1	1	1	1
PVOE	SRE • (XMM<1)	SRE • (XMM<2)	SRE • (XMM<3)	SRE • (XMM<4)
PVOV	A11	A10	A9	A8
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	–	–	–	–

Note: 1. 在 ATmega103 兼容模式下 XMM = 0。

Table 35. PC3..PC0 的第二功能 [1]

信号名称	**PC3/A11**	**PC2/A10**	**PC1/A9**	**PC0/A8**
PUOE	SRE • (XMM<5)	SRE • (XMM<6)	SRE • (XMM<7)	SRE • (XMM<7)
PUOV	0	0	0	0
DDOE	SRE • (XMM<5)	SRE • (XMM<6)	SRE • (XMM<7)	SRE • (XMM<7)
DDOV	1	1	1	1
PVOE	SRE • (XMM<5)	SRE • (XMM<6)	SRE • (XMM<7)	SRE • (XMM<7)
PVOV	A11	A10	A9	A8
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	–	–	–	–

Note: 1. 在 ATmega103 兼容模式下 XMM = 0。

端口 D 的第二功能

端口 D 的第二功能列于 Table 36。

Table 36. 端口 D 的第二功能

端口引脚	第二功能
PD7	T2 (T/C2 的时钟输入)
PD6	T1 (T/C1 的时钟输入)
PD5	XCK1[1] (USART1 的外部时钟输入 / 输出)
PD4	ICP1 (T/C1 输入捕捉的触发引脚)
PD3	INT3/TXD1[1] (外部中断 3 的输入引脚，或是 UART1 发送引脚)
PD2	INT2/RXD1[1] (外部中断 2 的输入引脚，或是 UART1 接收引脚)
PD1	INT1/SDA[1] (外部中断 1 的输入引脚，或是 TWI 的串行数据)
PD0	INT0/SCL[1] (外部中断 0 的输入引脚，或是 TWI 的串行时钟)

Note: 1. ATmega103 兼容模式没有 XCK1、 TXD1、 RXD1、 SDA 和 SCL。

第二功能配置如下：

- **T2 – 端口 D, Bit 7**

T2 为 T/C2 的计数输入源。

- **T1 – 端口 D, Bit 6**

T1 为 T/C1 的计数输入源。

- **XCK1 – 端口 D, Bit 4**

XCK1为USART1的外部时钟。数据方向寄存器 (DDD4)控制时钟为输入(DDD4 为'0')还是输出 (DDD4 为 '1')。只有当 USART1 工作于同步模式时 XCK1 才有效。



- **ICP1 – 端口 D, Bit 4**

ICP1 – 输入捕捉引脚 1：PD4 可以作为 T/C1 的输入捕捉引脚。

- **INT3/TXD1 – 端口 D, Bit 3**

INT3，外部中断源 3：PD3 可以作为 MCU 的外部中断源。

TXD1 是 USART1 的数据发送引脚。当使能了 USART1 的发送器后，这个引脚被强制设置为输出，此时 DDD3 不起作用。

- **INT2/RXD1 – 端口 D, Bit 2**

INT2，外部中断源 2。 PD2 可以作为 MCU 的外部中断源。

RXD1 是 USART1 的数据接收引脚。当使能了 USART1 的接收器后，这个引脚被强制设置为输出，此时 DDD2 不起作用。但是 PORTD2 仍然控制上拉电阻。

- **INT1/SDA – 端口 D, Bit 1**

INT1，外部中断源 1。 PD1 可以作为 MCU 的外部中断源。

SDA，两线接口的数据引脚：当寄存器 TWCR 的 TWEN 置位时使能两线接口。引脚 PD1 与端口脱离开而成为两线接口的串行数据 I/O 引脚。此时，引脚配置一个尖峰滤波器以抑制 50ns 以下的尖峰信号，而引脚由具有斜率限制功能的开漏驱动器驱动。

- **INT0/SCL – 端口 D, Bit 0**

INT0 为外部中断源 0。 PD0 可以作为 MCU 的外部中断源。

SCL，两线接口的时钟：当寄存器 TWCR 的 TWEN 置位时使能两线接口。引脚 PD0 与端口脱离开而成为两线接口的串行数据时钟 I/O 引脚。此时，引脚配置一个尖峰滤波器以抑制 50ns 以下的尖峰信号，而引脚由具有斜率限制功能的开漏驱动器驱动。

Table 37 和 Table 38 将端口 D 的第二功能与 P 66Figure 33 的重载信号关联在了一起。

Table 37. PD7..PD4 的第二功能

信号名称	PD7/T2	PD6/T1	PD5/XCK1	PD4/ICP1
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	UMSEL1	0
PVOV	0	0	XCK1 输出	0
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	T2 输入	T1 输入	XCK1 输入	ICP1 输入
AIO	–	–	–	–

Table 38. PD3..PD0 的第二功能 (1)

信号名称	PD3/INT3/TXD1	PD2/INT2/RXD1	PD1/INT1/SDA	PD0/INT0/SCL
PUOE	TXEN1	RXEN1	TWEN	TWEN
PUOV	0	PORTD2 • $\overline{\text{PUD}}$	PORTD1 • $\overline{\text{PUD}}$	PORTD0 • $\overline{\text{PUD}}$
DDOE	TXEN1	RXEN1	TWEN	TWEN
DDOV	1	0	SDA_OUT	SCL_OUT
PVOE	TXEN1	0	TWEN	TWEN
PVOV	TXD1	0	0	0
DIEOE	INT3 使能	INT2 使能	INT1 使能	INT0 使能
DIEOV	1	1	1	1
DI	INT3 输入	INT2 输入 /RXD1	INT1 输入	INT0 输入
AIO	–	–	SDA 输入	SCL 输入

Note: 1. 使能后 ,两线串行接口为 PD0 和 PD1 引入信号斜率控制。但是在本表中没有说明这一点。另外，尖峰滤波器连接与 AIO 输出和 TWI 模块的数字逻辑之间。

端口 E 的第二功能

端口 E 的第二功能列于 Table 39。

Table 39. 端口 E 的第二功能

引脚	第二功能
PE7	INT7/IC3(1) (外部中断 7 的输入引脚，或是 T/C3 输入捕捉的触发引脚)
PE6	INT6/ T3(1) (外部中断 6 的输入引脚，或是 T/C3 的时钟输入)
PE5	INT5/OC3C(1) (外部中断 5 的输入引脚 ,或是 T/C3 的输出比较和 PWM 输出 C 引脚)
PE4	INT4/OC3B(1) (外部中断 4 的输入引脚 ,或是 T/C3 的输出比较和 PWM 输出 B 引脚)
PE3	AIN1/OC3A (1) (模拟比较器负输入端，或是 T/C3 的输出比较和 PWM 输出 A 引脚)
PE2	AIN0/XCK0(1) (模拟比较器正输入端，或是 USART0 的外部输入 / 输出时钟)
PE1	PDO/TXD0 (编程数据输出，或是 USART0 的发送引脚)
PE0	PDI/RXD0 (编程数据输出，或是 USART0 的接收引脚)

Note: 1. ATmega103 兼容模式下没有 ICP3、T3、OC3C、OC3B、OC3B、OC3A 和 XCK0。

- **INT7/ICP3 – 端口 E, Bit 7**

INT7，外部中断源 7：PE7 可以作为 MCU 的外部中断源。

ICP3 – 输入捕捉引脚 3：PE7 可以作为 T/C3 的输入捕捉引脚。

- **INT6/T3 – 端口 E, Bit 6**

INT6，外部中断源 6：PE6 可以作为外部中断源。

T3， T/C3 的计数输入源。

- **INT5/OC3C – 端口 E, Bit 5**

INT5，外部中断源 5：PE5 可以作为外部中断源。

OC3C ,输出比较匹配 C 的输出: PE5 可以作为 T/C3 输出比较 C 的输出引脚。此时需要置位 DDE5 以将其配置为输出。 OC3C 还可以作为 PWM 模式的输出。

- **INT4/OC3B – 端口 E, Bit 4**



INT4，外部中断源 4：PE4 可以作为外部中断源。

OC3B，输出比较匹配 B 的输出。PE4 可以作为 T/C3 输出比较 B 的输出引脚。此时需要置位 DDE4 以将其配置为输出。 OC3B 还可以作为 PWM 模式的输出。

- **AIN1/OC3A – 端口 E, Bit 3**

AIN1 – 模拟比较器负输入端。

OC3A，输出比较匹配 A 的输出。PE3 可以作为 T/C3 输出比较 A 的输出引脚。此时需要置位 DDE3 以将其配置为输出。 OC3A 还可以作为 PWM 模式的输出。

- **AIN0/XCK0 – 端口 E, Bit 2**

AIN0 – 模拟比较器正输入端。

XCK0，USART0 的外部时钟。数据方向寄存器的 DDE2 控制这个时钟是输入时钟 (DDE2 为 '0') 还是输出时钟 (DDE2 为 '1')。只有当 USART0 工作于同步模式时 XCK0 才会生效。

- **PDO/TXD0 – 端口 E, Bit 1**

PDO， SPI 串行编程的数据输出。在串行下载程序时这个引脚用来输出数据。

TXD0， UART0 发送引脚。

- **PDI/RXD0 – 端口 E, Bit 0**

PDI， SPI 串行编程的数据输入。在串行下载程序时这个引脚用来输入数据。

RXD0，USART0接收引脚。当使能USART0接收器后这个引脚配置为输入而不管DDRE0 的设置如何。 PORTE0 仍然控制着上拉电阻的使能。

Table 40 和 Table 41 将端口 E 的第二功能与 P 66Figure 33 的重载信号关联在了一起。

Table 40. PE7..PE4 的第二功能

信号名称	PE7/INT7/ICP3	PE6/INT6/T3	PE5/INT5/OC3C	PE4/INT4/OC3B
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	OC3C 使能	OC3B 使能
PVOV	0	0	OC3C	OC3B
DIEOE	INT7 使能	INT6 使能	INT5 使能	INT4 使能
DIEOV	1	1	1	1
DI	INT7 输入 /ICP3 输入	INT7 输入 /T3 输入	INT5 输入	INT4 输入
AIO	–	–	–	–

Table 41. PE3..PE0 的第二功能

信号名称	PE3/AIN1/OC3A	PE2/AIN0/XCK0	PE1/PDO/TXD0	PE0/PDI/RXD0
PUOE	0	0	TXEN0	RXEN0
PUOV	0	0	0	PORTE0 • $\overline{PUD}$
DDOE	0	0	TXEN0	RXEN0
DDOV	0	0	1	0
PVOE	OC3B 使能	UMSEL0	TXEN0	0
PVOV	OC3B	XCK0 输出	TXD0	0
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	0	XCK0 输入	–	RXD0
AIO	AIN1 输入	AIN0 输入	–	–

端口 F 的第二功能

如 Table 42 所示，端口 F 的第二功能是 ADC 输入。如果端口 F 的一些引脚配置为输出，则很重要的一点是在 AD 转换过程中不要改变输出引脚的电平，否则可能会毁坏转换结果。在 ATmega103 兼容模式下端口 F 只能作为输入。若使能了 JTAG 接口，则即使在复位阶段 PF7(TDI)、 PF5(TMS) 和 PF4(TCK) 的上拉电阻仍然有效。

Table 42. 端口 F 的第二功能

端口引脚	第二功能
PF7	ADC7/TDI (ADC 输入通道 7，或是 JTAG 测试数据输入引脚)
PF6	ADC6/TDO (AD 输入通道 6，或是 JTAG 测试数据输出引脚)
PF5	ADC5/TMS (ADC 输入通道 5 ，或是 JTAG 测试模式选择引脚)
PF4	ADC4/TCK (ADC 输入通道 4，或是 JTAG 测试时钟)
PF3	ADC3 (ADC 输入通道 3)
PF2	ADC2 (ADC 输入通道 2)
PF1	ADC1 (ADC 输入通道 1)
PF0	ADC0 (ADC 输入通道 0)

- **TDI, ADC7 – 端口 F, Bit 7**

ADC7，模数转换器通道 7。

TDI，JTAG 测试数据输入引脚: 将要移入指令寄存器或数据寄存器 (扫描链) 的串行输入数据。使能 JTAG 接口之后这个引脚不能再用作普通 I/O 口。

- **TDO, ADC6 – 端口 F, Bit 6**

ADC6，模数转换器通道 6。

TDO，JTAG 测试数据输出引脚 : 将要移入指令寄存器或数据寄存器的串行输出数据。使能 JTAG 接口之后这个引脚不能再用作普通 I/O 口。

除 TAP 状态外 TDO 引脚为三态。

- **TMS, ADC5 – 端口 F, Bit 5**

ADC5，模数转换器通道 5。



TMS ,JTAG 测试模式选择引脚。这个引脚用于 TAP 控制器状态机 的定位。使能 JTAG 接口之后这个引脚不能再用作普通 I/O 口。

- **TCK, ADC4 – 端口 F, Bit 4**

ADC4 ，模数转换器通道 4。

TCK ,JTAG 测试时钟 : JTAG 的操作相对 TCK 是同步的。使能 JTAG 接口之后这个引脚不能再用作普通 I/O 口。

- **ADC3 – ADC0 – 端口 F, Bit 3..0**

模数转换器通道 3..0。

Table 43. PF7..PF4 的第二功能

信号名称	PF7/ADC7/TDI	PF6/ADC6/TDO	PF5/ADC5/TMS	PF4/ADC4/TCK
PUOE	JTAGEN	JTAGEN	JTAGEN	JTAGEN
PUOV	1	0	1	1
DDOE	JTAGEN	JTAGEN	JTAGEN	JTAGEN
DDOV	0	SHIFT_IR + SHIFT_DR	0	0
PVOE	0	JTAGEN	0	0
PVOV	0	TDO	0	0
DIEOE	JTAGEN	JTAGEN	JTAGEN	JTAGEN
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	TDI/ADC7 输入	ADC6 输入	TMS/ADC5 输入	TCKADC4 输入

Table 44. PF3..PF0 第二功能的重载信号

信号名称	PF3/ADC3	PF2/ADC2	PF1/ADC1	PF0/ADC0
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	0	0	0	0
PVOV	0	0	0	0
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	ADC3 输入	ADC2 输入	ADC1 输入	ADC0 输入

端口 G 的第二功能

在 ATmega103 兼容模式下端口 G 只具有下面描述的第二功能，而不能用作通用数字端口。第二功能介绍如下：

Table 45. 端口 G 的第二功能

端口引脚	第二功能
PG4	TOSC1 (RTC 振荡器， T/C0)
PG3	TOSC2 (RTC 振荡器， T/C0)
PG2	ALE (外部存储器地址锁存使能信号)
PG1	$\overline{\text{RD}}$ (外部存储器读信号)
PG0	$\overline{\text{WR}}$ (外部存储器写信号)

- **TOSC1 – 端口 G, Bit 4**

TOSC1，定时器振荡器引脚1：当寄存器ASSR的AS0置位时使能T/C0的异步时钟。PG4从端口上脱离，成为反向振荡器放大器的输入。此时可以外接晶体振荡器，同时不能用作I/O。

- **TOSC2 – 端口 G, Bit 3**

TOSC2，定时器振荡器引脚2：当寄存器ASSR的AS0置位时使能T/C0的异步时钟。PG3从端口上脱离，成为反向振荡器放大器的输入。此时可以外接晶体振荡器，同时不能用作I/O。

- **ALE – 端口 G, Bit 2**

ALE 为外部存储器地址锁存使能信号。

- **$\overline{\text{RD}}$ – 端口 G, Bit 1**

$\overline{\text{RD}}$ 是外部存储器读控制信号。

- **$\overline{\text{WR}}$ – 端口 G, Bit 0**

$\overline{\text{WR}}$ 是外部存储器写控制信号。

Table 46 和 Table 47 将端口 E 的第二功能与 P 66Figure 33 的重载信号关联在了一起。

Table 46. PG4..PG1 的第二功能

信号名称	PG4/TOSC1	PG3/TOSC2	PG2/ALE	PG1/$\overline{\text{RD}}$
PUOE	AS0	AS0	SRE	SRE
PUOV	0	0	0	0
DDOE	AS0	AS0	SRE	SRE
DDOV	0	0	1	1
PVOE	0	0	SRE	SRE
PVOV	0	0	ALE	RD
DIEOE	AS0	AS0	0	0
DIEOV	0	0	0	0
DI	–	–	–	–
AIO	T/C0 OSC 输入	T/C0 OSC 输出	–	–



Table 47. PG0 的第二功能

信号名称	PG0/$\overline{\text{WR}}$
PUOE	SRE
PUOV	0
DDOE	SRE
DDOV	1
PVOE	SRE
PVOV	WR
DIEOE	0
DIEOV	0
DI	–
AIO	–

I/O 端口寄存器的说明

端口 A 数据寄存器 - PORTA

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 A 数据方向寄存器 - DDRA

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 A 输入引脚地址 - PINA

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
读 / 写	R	R	R	R	R	R	R	R	
初始值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

端口 B 数据寄存器 - PORTB

Bit	7	6	5	4	3	2	1	0	
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 B 数据方向寄存器 - DDRB

Bit	7	6	5	4	3	2	1	0	
	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 B 输入引脚地址 - PINB

Bit	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
读 / 写	R	R	R	R	R	R	R	R	
初始值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

端口 C 数据寄存器 - PORTC

Bit	7	6	5	4	3	2	1	0	
	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 C 数据方向寄存器 - DDRC

Bit	7	6	5	4	3	2	1	0	
	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	



端口 C 输入引脚地址 - PINC

Bit	7	6	5	4	3	2	1	0	
	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
读 / 写	R	R	R	R	R	R	R	R	
初始值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

在 ATmega103 兼容模式下， DDRC 和 PINC 寄存器初始化为输出低电平。即使在没有时钟的情况下口线也将保持初始值。要注意的是，在 ATmega103 兼容模式下 DDRC 和是不可见的。为了保持 100% 的向后兼容，请不要在 ATmega103 兼容模式下访问这两个寄存器。

端口 D 数据寄存器 - PORTD

Bit	7	6	5	4	3	2	1	0	
	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 D 数据方向寄存器 - DDRD

Bit	7	6	5	4	3	2	1	0	
	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 D 输入引脚地址 - PIND

Bit	7	6	5	4	3	2	1	0	
	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
读 / 写	R	R	R	R	R	R	R	R	
初始值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

端口 E 数据寄存器 - PORTE

Bit	7	6	5	4	3	2	1	0	
	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0	PORTE
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 E 数据方向寄存器 - DDRE

Bit	7	6	5	4	3	2	1	0	
	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0	DDRE
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 E 输入引脚地址 - PINE

Bit	7	6	5	4	3	2	1	0	
	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0	PINF
读 / 写	R	R	R	R	R	R	R	R	
初始值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

端口 F 数据寄存器 - PORTF

Bit	7	6	5	4	3	2	1	0	
	PORTF7	PORTF6	PORTF5	PORTF4	PORTF3	PORTF2	PORTF1	PORTF0	PORTF
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 F 数据方向寄存器 - DDRF

Bit	7	6	5	4	3	2	1	0	
	DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0	DDRF
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 F 输入引脚地址 - PINF

Bit	7	6	5	4	3	2	1	0	
	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0	PINF
读 / 写	R	R	R	R	R	R	R	R	
初始值	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

在 ATmega103 兼容模式下 PORTF 和 DDRF 是不可见的，因为此时端口 F 只能作为输入引脚。

端口 G 数据寄存器 - PORTG

Bit	7	6	5	4	3	2	1	0	
	–	–	–	PORTG4	PORTG3	PORTG2	PORTG1	PORTG0	PORTG
读 / 写	R	R	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 G 数据方向寄存器 - DDRG

Bit	7	6	5	4	3	2	1	0	
	–	–	–	DDG4	DDG3	DDG2	DDG1	DDG0	DDRG
读 / 写	R	R	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

端口 G 输入引脚地址 - PING

Bit	7	6	5	4	3	2	1	0	
	–	–	–	PING4	PING3	PING2	PING1	PING0	PING
读 / 写	R	R	R	R	R	R	R	R	
初始值	0	0	0	N/A	N/A	N/A	N/A	N/A	

在 ATmega103 兼容模式下 PORTG、 DDRG 和 PING 是不可见的，只能用作第二功能，即 TOSC1、 TOSC2、 $\overline{\text{WR}}$、 $\overline{\text{RD}}$ 和 ALE)。

外部中断

外部中断通过引脚 INT7:0 触发。只要使能了中断，即使引脚 INT7:0 配置为输出，只要电平发生了合适的变化，中断也会触发。这个特点可以用来产生软件中断。通过设置外部中断控制寄存器 – EICRA (INT3:0) 和 EICRB (INT7:4)，中断可以由下降沿、上升沿，或者是低电平触发。当外部中断使能并且配置为电平触发，只要引脚电平为低，中断就会产生。若要求 INT7:4 在信号下降沿或上升沿触发， I/O 时钟必须工作，如 P 33“ 时钟系统及其分布 ” 说明的那样。 INT3:0 的中断条件检测则是异步的。也就是说，这些中断可以用来将器件从睡眠模式唤醒。在睡眠过程（除了空闲模式）中 I/O 时钟是停止的。

通过电平方式触发中断，从而将 MCU 从掉电模式唤醒时，要保证电平保持一定的时间，以降低 MCU 对噪声的敏感程度。电平以看门狗的频率检测两次。在 5.0V、 25°C 的条件下，看门狗的标称时钟周期为 1 µs。看门狗时钟受电压的影响，具体请参考 P 299“ 电气特性 ”。只要在采样过程中出现了合适的电平，或是信号持续到启动过程的末尾， MCU 就会唤醒。启动过程由熔丝位 SUT 决定，如 P 33“ 时钟系统及其分布 ” 所示。若信号出现于两次采样过程，但在启动过程结束之前就消失了， MCU 仍将唤醒，但不再会引发中断了。要求的电平必须保持足够长的时间以使 MCU 结束唤醒过程，然后触发电平中断。

外部中断控制寄存器 A - EICRA

Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

ATmega103 兼容模式不能访问这个寄存器，但是 INT3:0 的初始值定义为低电平中断。

- **Bits 7..0 – ISC31, ISC30 – ISC00, ISC00: 外部中断 3 - 0 敏感电平控制位**

外部中断 3 - 0 由引脚 INT3:0 激活，如果 SREG 寄存器的 I 标志和 EIMSK 寄存器相应的中断屏蔽位置位的话。触发方式如 Table 48 所示。INT3..INT0 的边沿触发方式是异步的。只要 INT3:0 引脚上产生宽度大于 Table 49 所示数据的脉冲就会引发中断。若选择了低电平中断，低电平必须保持到当前指令完成，然后才会产生中断。而且只要将引脚拉低，就会引发中断请求。改变 ISCn 时有可能发生中断。因此建议首先在寄存器 EIMSK 里清除相应的中断使能位 INTn，然后再改变 ISCn。最后，不要忘记在重新使能中断之前通过对 EIFR 寄存器的相应中断标志位 INTFn 写 '1' 使其清零。

Table 48. 中断敏感电平控制 (1)

ISCn1	ISCn0	说明
0	0	INTn 为低电平时产生中断请求
0	1	保留
1	0	INTn 的下降沿产生异步中断请求
1	1	INTn 的上升沿产生异步中断请求

Note: 1. n =3、 2、 1 或 0。
改变 ISCn1/ISCn0 时一定要先通过清零 EIMSK 寄存器的中断使能位来禁止中断。否则在改变 ISCn1/ISCn0 的过程中可能发生中断。

Table 49. 异步（外部）中断特性

符号	参数	条件	最小值	典型值	最大值	单位
t_{INT}	异步（外部）中断的最小脉冲宽度			50		ns

外部中断控制寄存器 B - EICRB

Bit	7	6	5	4	3	2	1	0	
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB

读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	0	0	0

- **Bits 7..0 – ISC71, ISC70 - ISC41, ISC40: 外部中断 7 - 4 敏感电平控制位**

外部中断 7 - 4 由引脚 INT7:4 激活，如果 SREG 寄存器的 I 标志和 EIMSK 寄存器相应的中断屏蔽位置位的话。触发方式如 Table 50 所示。检测信号跳变沿之前 MCU 首先对 INT7:4 引脚进行采样。如果选择了跳变沿中断或是电平变换中断 (上升沿和下降沿都将产生中断) , 只要信号持续时间大于一个时钟周期 , 中断就会发生；否则无法保证触发中断。要注意由于 XTAL 分频器的存在， CPU 时钟有可能比 XTAL 时钟慢。若选择了低电平中断，低电平必须保持到当前指令完成，然后才会产生中断。而且只要将引脚拉低，就会引发中断请求。

Table 50. 中断敏感电平控制 [1]

ISCn1	ISCn0	说明
0	0	INTn 为低电平时产生中断请求
0	1	INTn 引脚上任意的逻辑电平变换都将引发中断
1	0	只要两次采样发现 INTn 上发生了下降沿就会产生中断请求
1	1	只要两次采样发现 INTn 上发生了上升沿就会产生中断请求

Note: 1. n = 7、 6、 5 或 4。
改变 ISCn1/ISCn0 时一定要先通过清零 EIMSK 寄存器的中断使能位来禁止中断。否则在改变 ISCn1/ISCn0 的过程中可能发生中断。

外部中断屏蔽寄存器 - EIMSK

Bit	7	6	5	4	3	2	1	0	
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	IINT0	EIMSK
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bits 7..0 – INT7 – INT0: 外部中断请求 7 - 0 使能**

当 INT7 – INT0 为 '1'，而且状态寄存器 SREG 的 I 标志置位，相应的外部引脚中断就使能了。外部中断控制寄存器 – EICRA 和 EICRB 的中断敏感电平控制位决定中断是由上升沿、下降沿，还是电平触发的。只要使能，即使引脚被配置为输出，只要引脚电平发生了相应的变化，中断可将产生。据此可以实现软件中断。

外部中断标志寄存器 - EIFR

Bit	7	6	5	4	3	2	1	0	
	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	IINTF0	EIFR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bits 7..0 – INTF7 - INTF0: 外部中断标志 7 - 0**

INT7:0 引脚电平发生跳变时触发中断请求 , 并置位相应的中断标志 INTF7:0 。如果 SREG 的位 I 以及 EIMSK 寄存器相应的中断使能位为 '1'， MCU 既跳转到中断例程。中断例程执行时标志被硬件清零。此外，标志位也可以通过写入 '1' 的方式来清零。若 INT7:0 配置为电平触发，这些标志位总是为 '0'。在睡眠模式下，如果中断是禁止的，则这些引脚的输入缓冲器也是禁止的。这有可能产生逻辑电平的变化并置位 INTF3:0。更多信息请参考 P 65“ 数字输入使能和睡眠模式 ” 。





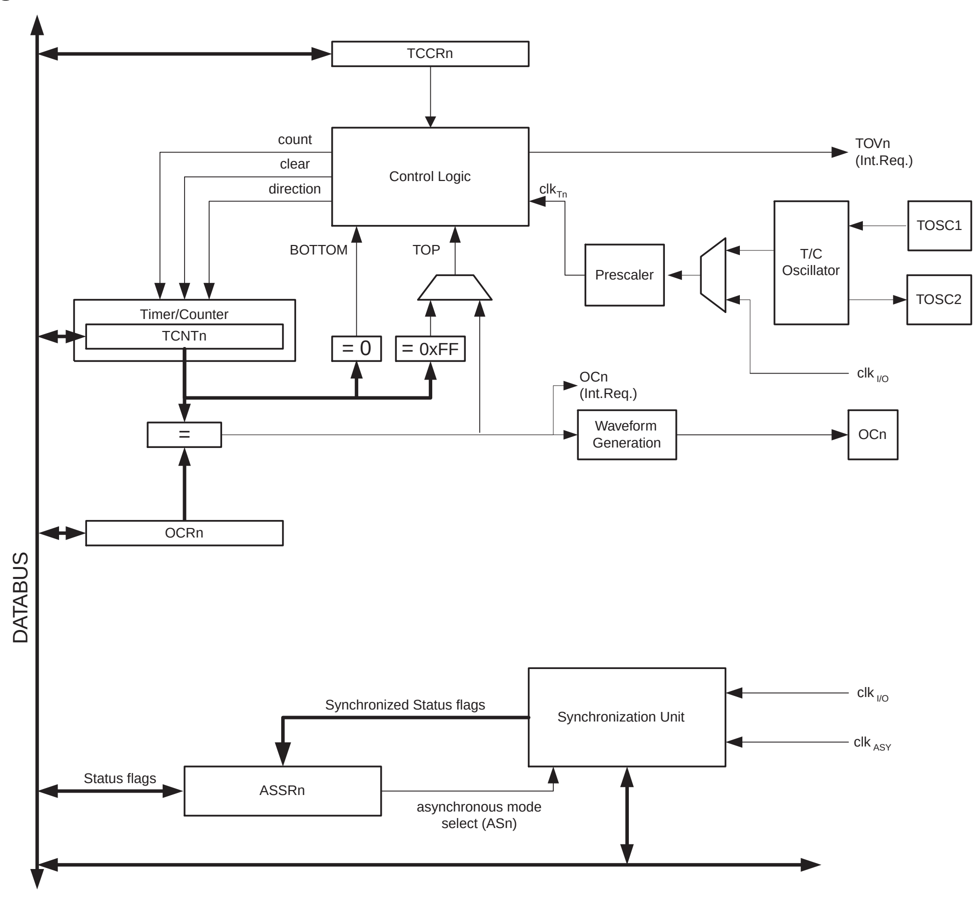
8 位定时器 / 计时器 0 - - PWM、异步操作

T/C0 是一个通用的，单通道 8 位定时器 / 计数器模块。其主要特点如下：
- **单通道计数器**
- **比较匹配发生时清除定时器 (自动加载)**
- **无毛刺的相位修正 PWM**
- **频率发生器**
- **10 位时钟预分频器**
- **溢出和比较匹配中断源 (TOV0 和 OCF0)**
- **允许外部 32kHz 钟振作为时钟**

综述

Figure 34为8位定时器/计数器的简化框图。实际引脚排放请参考 P 2"引脚配置" 。CPU可以访问的 I/O 寄存器，包括位和引脚，以粗体显示。I/O 寄存器和位的位置列于 P 94"8 位 T/C 寄存器说明 " 。

Figure 34. 8 位 T/C 方框图



寄存器

T/C(TCNT0)和输出比较寄存器(OCR0)为8位寄存器。中断请求信号位于定时器中断标志寄存器 TIFR。与定时器相关的所有中断都可以通过定时器中断屏蔽寄存器 TIMSK 单独进行屏蔽。由于 TIFR 和 TIMSK 寄存器由几个定时器单元共享，所以图中并没有表示出来。

T/C 可以通过预分频器由内部驱动，或者是由TOSC1/2时钟异步驱动。异步操作受异步状态寄存器 ASSR 控制。时钟选择模块则控制使用哪一个时钟源。没有选择时钟源时 T/C 不工作。时钟选择模块的输出记为定时器时钟 clk_{T0}。

双缓冲的输出比较寄存器 OCR0 一直与 T/C 的数值进行比较。比较的结果可用来产生PWM波，或在输出比较引脚OC0上产生变化频率的输出，如P 88 "输出比较单元" 说明的那样。比较匹配事件还将设置比较标志 OCF0。此标志可以用来产生输出比较中断请求。

定义

本文的许多寄存器及其各个位以通用的格式表示。小写的 "n" 取代了 T/C 的序号，在此即为 0。但是在写程序时要使用精确的格式 (例如使用 TCNT0 来访问 T/C0 计数器值)。

Table 51 的定义适用于全文。

Table 51. 定义

BOTTOM	计数器计到 0x00 时即达到 BOTTOM
MAX	计数器计到 0xFF (十进制的 255) 时即达到 MAX
TOP	计数器计到计数序列的最大值时即达到 TOP。 TOP 值可以分配为固定值 0xFF (MAX)，或是存储于寄存器 OCR0 里的数值，具体由工作模式确定

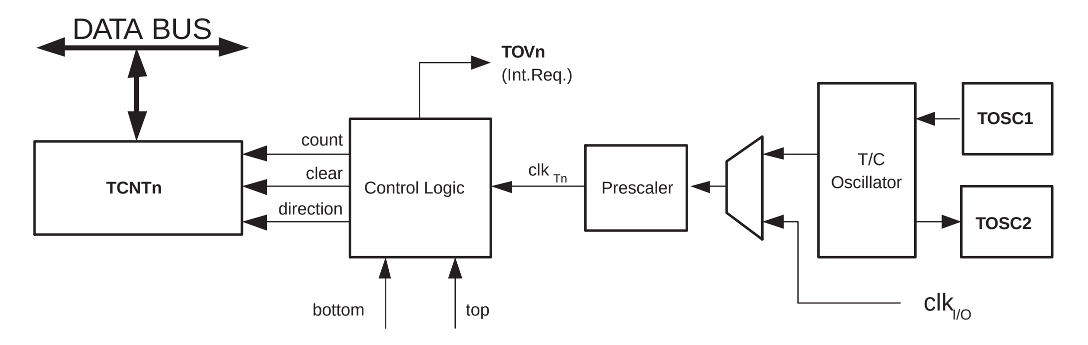
T/C 的时钟源

T/C 可以由内部同步时钟或外部异步时钟驱动。 clk_{T0} 的缺省设置为 MCU 时钟 $clk_{I/O}$。当 ASSR 寄存器的 AS0 置位时，时钟源取自连接于 TOSC1 和 TOSC2 的振荡器。详细的异步操作可以参考 P 97" 异步状态寄存器 - ASSR" 一节；详细的时钟源与预分频器的内容请参考 P 99" 定时器 / 计数器预分频器 " 。

计数器单元

8 位 T/C 的主要部分为可编程的双向计数单元。Figure 35 即为计数器和相关资源的方框图。

Figure 35. 计数器单元方框图



信号说明 (内部信号) :

count	使 TCNT0 加 1 或减 1。
direction	选择加操作或减操作。
clear	清除 TCNT0 (将所有的位清零)。
clk_{T0}	T/C 的时钟。
top	表示 TCNT0 已经达到了最大值。
bottom	表示 TCNT0 已经达到了最小值 (0)。

根据不同的工作模式，计数器针对每一个 clk_{T0} 实现清零、加一或减一操作。clk_{T0} 可以由内部时钟源或外部时钟源产生，具体由时钟选择位 CS02:0 确定。没有选择时钟源时 (CS02:0 = 0) 定时器停止。但是不管有没有 clk_{T0}，CPU 都可以访问 TCNT0。CPU 写操作比计数器其他操作 (清零、加减操作) 的优先级高。

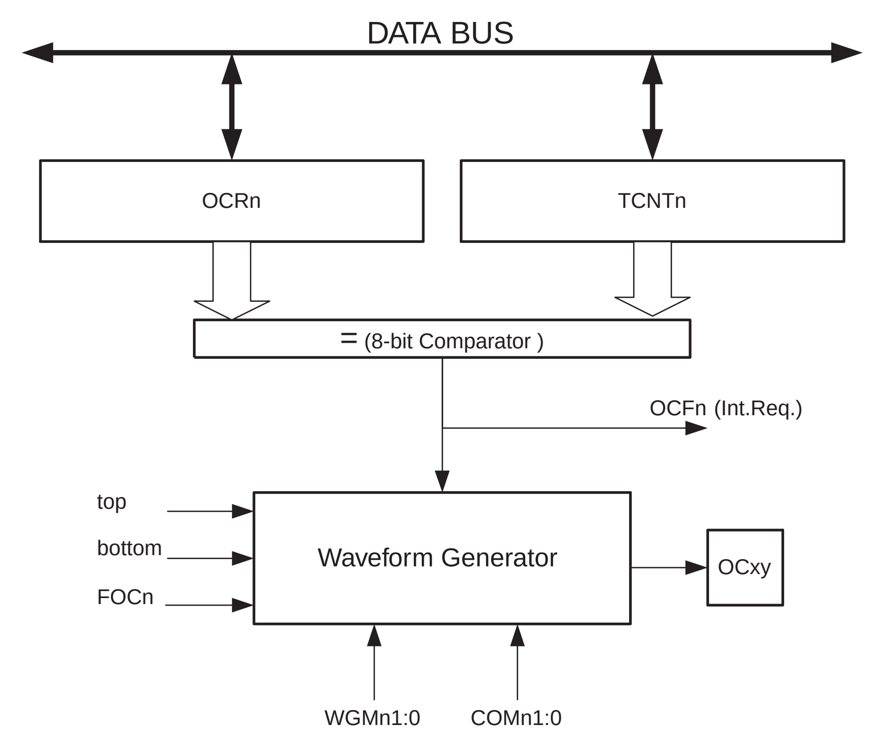
计数序列由 T/C 控制寄存器 (TCCR0) 的 WGM01 和 WGM00 决定。计数器计数行为与输出比较 OC0 的波形有紧密的关系。有关计数序列和波形产生的详细信息请参考 P 89" 工作模式 " 。

T/C溢出中断标志TOV0根据WGM01:0 设定的工作模式来设置。TOV0可以用于产生CPU中断。

输出比较单元

8 位比较器持续对 TCNT0 和输出比较匹配寄存器 OCR0 进行比较。一旦 TCNT0 等于 OCR0，比较器就给出匹配信号。在匹配发生的下一个定时器时钟周期里输出比较标志 OCF0置位。若OCIE0 = 1还将引发输出比较中断。执行中断将自动实现对OCF0的清零操作。也可以通过对这一位执行软件写'1'的操作来实行清零。根据由WGM01:0和COM01:0 设定的不同的工作模式，波形发生器利用匹配信号产生不同的波形。同时，波形发生器还利用 max 和 bottom 信号来处理极值条件下的特殊情况 (P 89“ 工作模式 ”)。Figure 36 为输出比较单元的方框图。

Figure 36. 输出比较单元方框图



使用 PWM 模式时 OCR0 寄存器为双缓冲寄存器；而在正常工作模式和匹配时清零模式双缓冲功能是禁止的。双缓冲可以将更新 OCR0 寄存器与 top 或 bottom 时刻同步起来，从而防止产生不对称的 PWM 脉冲，消除毛刺。

访问 OCR0 寄存器看起来很复杂，其实并不是。使能双缓冲功能时，CPU 访问的是 OCR0 缓冲寄存器；禁止双缓冲功能时 CPU 访问的则是 OCR0 本身。

强制输出比较

工作于非 PWM 模式时，可以通过对强制输出比较位 FOC0 写 '1' 的方式来产生比较匹配。强制比较匹配不会置位 OCF0 标志，也不会重载 / 清零定时器，但是 OC0 引脚将被更新，好象真的发生了比较匹配一样 (COM01:0 决定 OC0 是置位、清零，还是交替变化)。

写 TCNT0 操作阻止比较匹配

CPU 对 TCNT0 寄存器的写操作会阻止比较匹配的发生。这个特性可以用来将 OCR0 初始化为与 TCNT0 相同的数值而不触发中断。

使用输出比较单元

由于在任意模式下写 TCNT0 都将在下一个定时器时钟周期里阻止比较匹配，在使用输出比较时改变TCNT0就会有风险，不管T/C是否在运行。若写如TCNT0的数值等于OCR0，比较匹配就被忽略了，造成不正确的波形发生结果。类似地，在计数器进行降序计数时不要对 TCNT0 写入 BOTTOM。

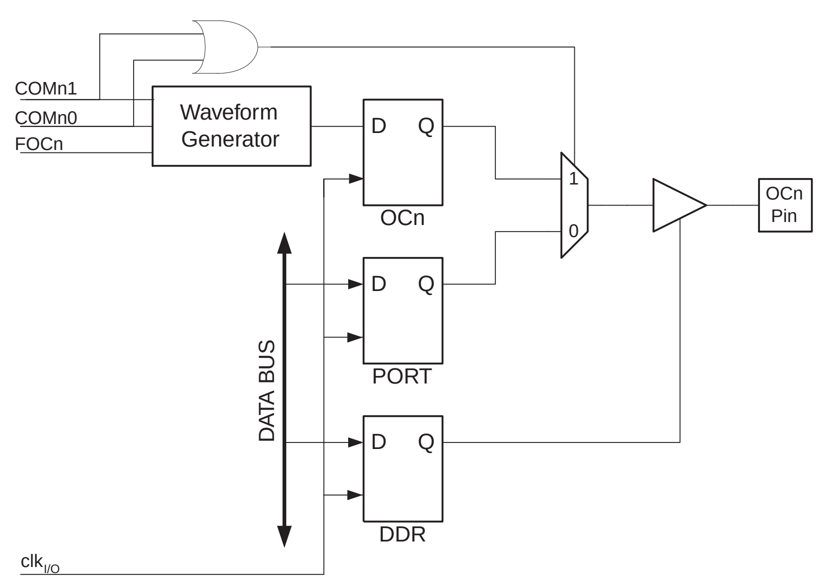
OC0 的设置应该在设置数据方向寄存器之前完成。最简单的设置 OC0 的方法是在普通模式下利用强制输出比较 FOC0。即使在改变波形发生模式时 OC0 寄存器也会一直保持它的数值。

COM01:0 和比较数据都不是双缓冲。 COM01:0 的改变将立即生效。

比较匹配输出单元

比较匹配模式控制位 COM01:0 具有双重功能。波形发生器利用 COM01:0 来确定下一次比较匹配发生时的输出比较 OC0 状态； COM01:0 还控制 OC0 引脚的输出源。Figure 37 为受 COM01:0 设置影响的逻辑的简化原理图。 I/O 寄存器、 I/O 位和 I/O 引脚以粗体表示。图中只给出了受 COM01:0 影响的通用 I/O 端口控制寄存器 (DDR 和 PORT)。谈及 OC0 状态时指的是内部 OC0 寄存器，而不是 OC0 引脚。

Figure 37. 比较匹配输出单元原理图



只要 COM01:0 的一个或两个置位，波形发生器的输出比较 OC0 功能就会重载通用 I/O 口功能。但是 OC0 引脚的方向仍旧受控于数据方向寄存器 (DDR)。在能够从 OC0 引脚输出有效信号之前必须通过数据方向寄存器的 DDR_OC0 位将此引脚设置为输出。功能重载与波形发生器的工作模式无关。

输出比较逻辑的设计允许 OC0 状态在输出之前首先初始化。要注意某些 COM01:0 设置是保留的，如 P 94 “8 位 T/C 寄存器说明 ” 所示。

比较输出模式和波形产生

波形发生器使用 COM01:0 的方式在普通、 CTC 和 PWM 三种模式下有所区别。对于所有的模式，设置 COM01:0 = 0 表明比较匹配发生时波形发生器无需操作 OC0 寄存器。非PWM模式的比较输出请参见 P 95Table 53 ；快速PWM的比较输出于P 95Table 54 ；相位修正 PWM 的比较输出于 P 96Table 55 。

改变 COM01:0 将影响写入数据后的第一次比较匹配。对于非 PWM 模式，可以通过使用 FOC0 来强制立即产生效果。

工作模式

工作模式，即 T/C 和输出比较引脚的行为，由波形发生模式 (WGM01:0) 及比较输出模式 (COM01:0) 的控制位决定。比较输出模式对计数序列没有影响，而波形产生模式对计数序列则有影响。 COM01:0 控制 PWM 输出是否反极性。非 PWM 模式时 COM01:0 控制输出是否应该在比较匹配发生时置位、清零，或是电平取反 (P 89 “ 比较匹配输出单元 ”)。

具体的时序信息请参考 P 93“T/C 时序图 ” 。

普通模式

普通模式 (WGM01:0 = 0) 为最简单的工作模式。在此模式下计数器不停地累加。计到最大值后 (TOP = 0xFF) 计数器简单地返回到最小值 0x00 重新开始。在 TCNT0 为零的同一个定时器时钟里 T/C 溢出标志 TOV0 置位。此时 TOV0 有点象第 9 位，只是只能置位，不会清零。但由于定时器中断例程能够自动清零 TOV0，因此可以通过软件提高定时器的分辨率。在普通模式下没有什么需要特殊考虑的，用户可以随时写入新的计数器数值。

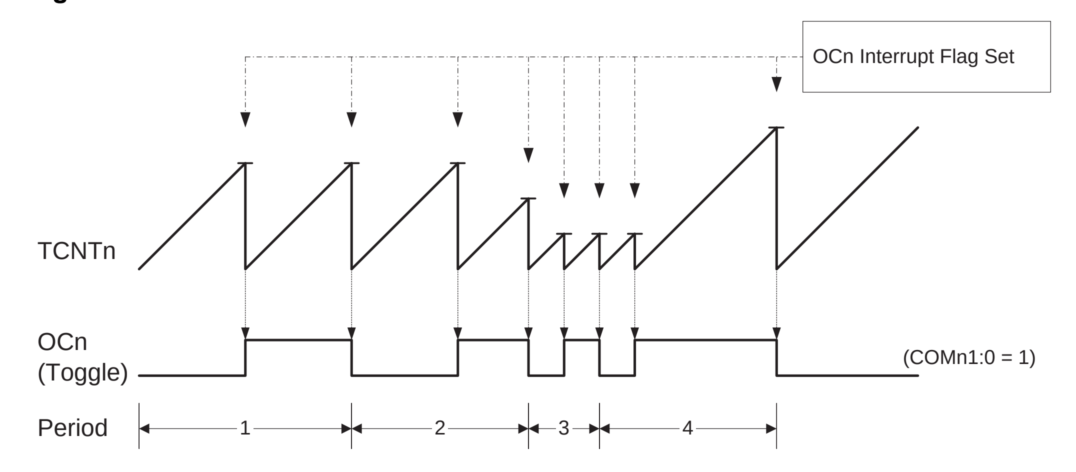
输出比较单元可以用来产生中断。不推荐在普通模式下利用输出比较产生波形，因为会占用太多的 CPU 时间。

CTC(比较匹配时清除定时器) 模式

在 CTC 模式 (WGM01:0 = 2) 里 OCR0 寄存器用于调节计数器的分辨率。当计数器的数值 TCNT0 等于 OCR0 时计数器清零。OCR0 定义了计数器的 top 值，亦即计数器的分辨率。这个模式可以在极大程度上控制比较匹配输出的频率，也简化了外部事件计数的操作。

CTC模式的时序图为 Figure 38。计数器数值TCNT0一直增加直到TCNT0与OCR0匹配，然后 TCNT0 清零。

Figure 38. CTC 模式的时序图



利用 OCF0 标志可以在计数器数值达到 TOP 即产生中断。若使能了中断，可以利用中断例程来更新 TOP 的数值。由于 CTC 模式没有双缓冲功能，在计数器以无预分频器或很低的预分频器工作的时候将 TOP 更改为接近 BOTTOM 的数值时要小心。如果写入 OCR0 的数值小于当前 TCNT0 的数值，计数器将丢失一次比较匹配。在下一次比较匹配发生之前，计数器不得不先计数到最大值 0xFF，然后再从 0x00 开始计数到 OCR0。

通过设置 COM01:0 = 1，可以在 CTC 模式下将比较输出模式设置为交替方式。只要比较匹配发生，OC0 的输出电平就取反。在期望获得 OC0 输出之前，首先要将其端口设置为输出。波形发生器能够产生的最大频率为 $f_{OC0} = f_{clk_I/O}/2$ (OCR0 = 0x00)。频率由如下公式确定：

$$f_{OCn} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRn)}$$

变量 N 代表预分频因子 (1、 8、 32、 64、 128、 256 或 1024)。

在普通模式下， TOV0 标志的置位发生在计数器从 MAX 变为 0x00 的定时器时钟周期。

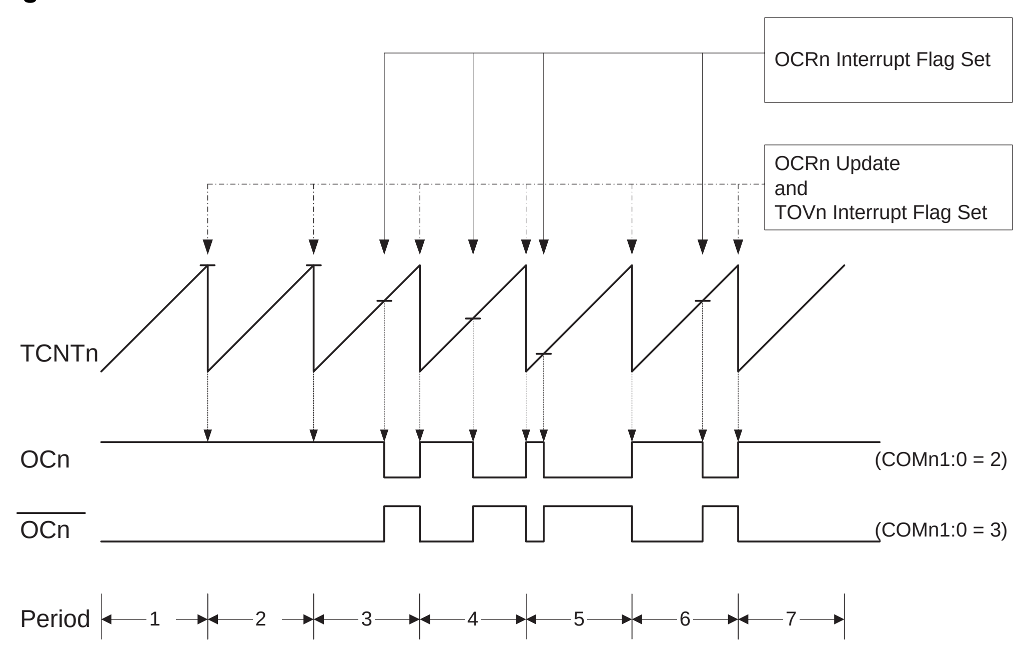
快速 PWM 模式

快速 PWM 模式 (WGM01:0 = 3) 可用来产生高频的 PWM 波形。快速 PWM 模式与其他 PWM模式的不同之处是其三角波工作方式(其他PWM方式为等腰三角形方式)。计数器从 BOTTOM 计到 MAX，然后立即回到 BOTTOM 重新开始。对于普通的比较输出模式，输出比较引脚 OC0 在 TCNT0 与 OCR0 匹配时清零，在 BOTTOM 时置位；对于反向比较输出模式， OC0 的动作正好相反。由于使用了单边斜波模式，快速 PWM 模式的工作频率比使用双斜波的相位修正 PWM 模式高一倍。此高频操作特性使得快速 PWM 模式十分适

合于功率调节，整流和 DAC 应用。高频可以减小外部元器件 (电感，电容) 的物理尺寸，从而降低系统成本。

工作于快速 PWM 模式时，计数器的数值一直增加到 MAX，然后在后面的一个时钟周期清零。具体的时序图为 Figure 39。图中柱状的 TCNT0 表示这是单边斜波操作。方框图同时包含了普通的 PWM 输出以及方向 PWM 输出。 TCNT0 斜波上的短水平线表示 OCR0 和 TCNT0 的匹配比较。

Figure 39. 快速 PWM 模式时序图



计时器数值达到 Max 时 T/C 溢出标志 TOV0 置位。如果中断使能，中断例程可用来更新比较值。

工作于快速 PWM 模式时，比较单元可以在 OC0 引脚上输出 PWM 波形。设置 COM01:0 为 2 可以产生普通的 PWM 信号；为 3 则可以产生反向 PWM 波形。(参见 P 95Table 54)。要想真正输出信号还必须将OC0的数据方向设置为输出。产生PWM波形的机理是OC0 寄存器在 OCR0 与 TCNT0 匹配时置位 (或清零)，以及在计数器清零 (从 MAX 变为 BOTTOM) 的那一个定时器时钟周期清零 (或置位)。

输出的 PWM 频率可以通过如下公式计算得到：

$$f_{OCnPWM} = \frac{f_{clk_I/O}}{N \cdot 256}$$

变量 N 代表分频因子 (1、 8、 32、 64、 128、 256 或 1024)。

OCR0 寄存器为极限值时表示快速 PWM 模式的一些特殊情况。若 OCR0 等于 BOTTOM，输出为出现在第 MAX+1 个定时器时钟周期的窄脉冲； OCR0 为 MAX 时，根据 COM01:0 的设定，输出恒为高电平或低电平。

通过设定 OC0 在比较匹配时进行逻辑电平取反 (COM01:0 = 1)，可以得到占空比为 50% 的周期信号。信号的最高频率为 $f_{oc0} = f_{clk_I/O}/2$，此时 OCR0 为 0。这个特性类似于 CTC 模式下的 OC0 取反操作，不同之处在于快速 PWM 模式具有双缓冲。



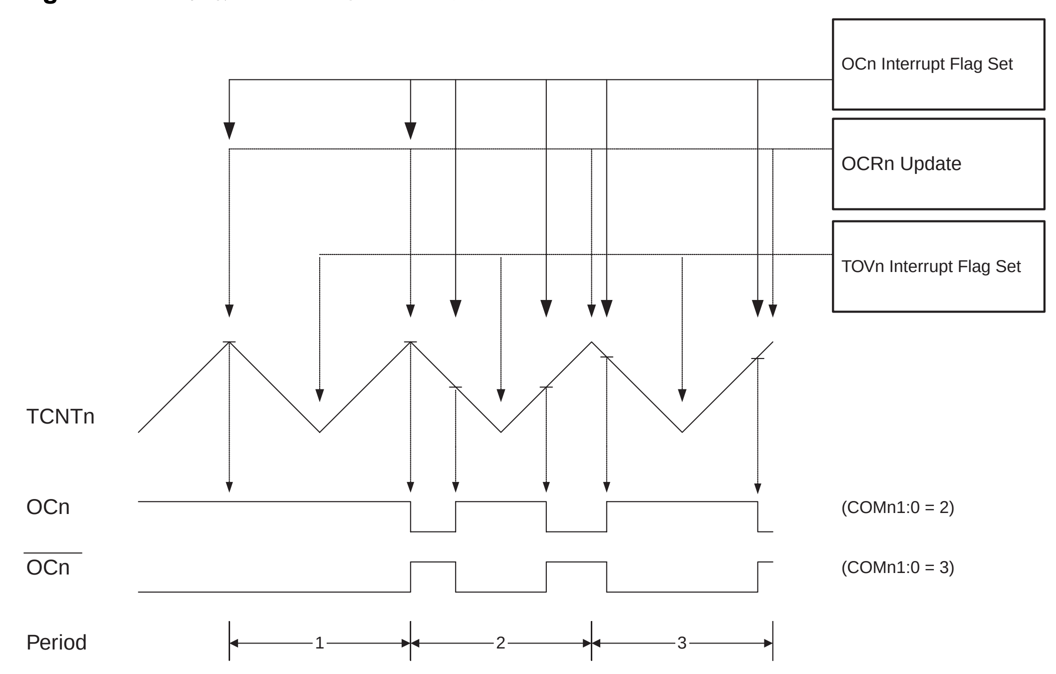


相位修正 PWM 模式

相位修正 PWM 模式 (WGM01:0 = 1) 为用户提供了一个获得高精度相位修正 PWM 波形的方法。此模式基于双斜线操作。计时器重复地从 BOTTOM 计到 MAX，然后又从 MAX 倒退回到BOTTOM。在一般的比较输出模式下，当计时器往MAX计数时若发生了TCNT0 于OCR0的匹配，OC0将清零为低电平；而在计时器往BOTTOM计数时若发生了TCNT0 于 OCR0 的匹配， OC0 将置位为高电平。工作于反向输出比较时则正好相反。与单斜线操作相比，双斜线操作可获得的最大频率要小。但由于其对称的特性，十分适合于电机控制。

相位修正 PWM 模式的 PWM 精度固定为 8 比特。计时器不断地累加直到 Max，然后开始减计数。在一个定时器时钟周期里 TCNT0 的值等于 MAX。时序图可参见 Figure 40。图中 TCNT0 的数值用柱状图表示，以说明双斜线操作。本图同时说明了普通 PWM 的输出和反向 PWM 的输出。 TCNT0 斜线上的小横条表示 OCR0 和 TCNT0 的匹配。

Figure 40. 相位修正 PWM 模式的时序图



当计时器达到 BOTTOM 时 T/C 溢出标志位 TOV0 置位。此标志位可用来产生中断。

工作于相位修正 PWM 模式时，比较单元可以在 OC0 引脚产生 PWM 波形：将 COM01:0 设置为 2 产生普通相位的 PWM，设置 COM01:0 为 3 产生反向 PWM 信号 (参见 P 96Table 55)。实际的 OC0 数值只有在端口设置为输出时才可以在引脚上出现。OCR0 和 TCNT0 比较匹配发生时 OC0 寄存器将产生相应的清零或置位操作，从而产生 PWM 波形。工作于相位修正模式时 PWM 频率可由下式公式获得：

$$f_{OCnPCPWM} = \frac{f_{clk_I/O}}{N \cdot 510}$$

变量 N 表示预分频因子 (1、 8、 32、 64、 128、 256 或 1024)。

OCR0寄存器处于极值代表了相位修正PWM模式的一些特殊情况。在普通PWM模式下，若 OCR0 等于 BOTTOM，输出一直保持为低电平；若 OCR0 等于 MAX，则输出保持为高电平。反向 PWM 模式则正好相反。

Figure 40 中的第二个时钟周期中，即使没有比较匹配，OCn 电平也会由高变低。这样保证关于 BOTTOM 对称。在以下两种情况下没有比较匹配时电平发生变化：

- 如 Figure 40 中 OCR0A 值由 MAX 改变。降序比较匹配时，当 OCR0A 值为 MAX，OCn 引脚的值也一样；为保证关于 BOTTOM 对称，在升序比较匹配时，OCn 值为 MAX 时电平变化。
- 定时器开始计数的值大于 OCR0A 中的值，因此少一次比较匹配，且在达到上限时 OCn 改变。

T/C 时序图

Figure 41 和 Figure 42 给出了定时器 / 计数器 (T/C) 的时序。T/C 为同步电路，亦即其时钟 clk_{T0} 表示为时钟使能信号。图中给出接近 MAX 值的计数序列。Figure 43 和 Figure 44 给出了预分频器使能时的时序。

下面图中给出同步模式下的 T/C 状态，clk_{T0} 为时钟使能信号。在异步模式下，$clk_{I/O}$ 由 T/C 振荡器时钟所取代。图中还包含何时设置中断标志的信息。Figure 41 包含了 T/C 的基本时序。图中给出除相位修正 PWM 模式外的接近 MAX 值的计数序列。

Figure 41. T/C 时序图，无预分频器

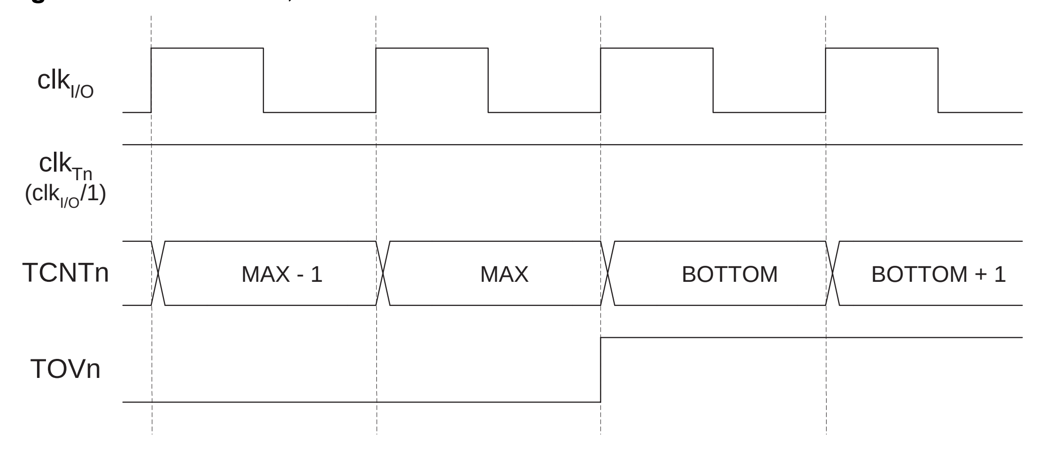


Figure 42 给出相同的计时数据，但预分频器使能。

Figure 42. T/C 时序图，预分频器为 $f_{clk_I/O}/8$

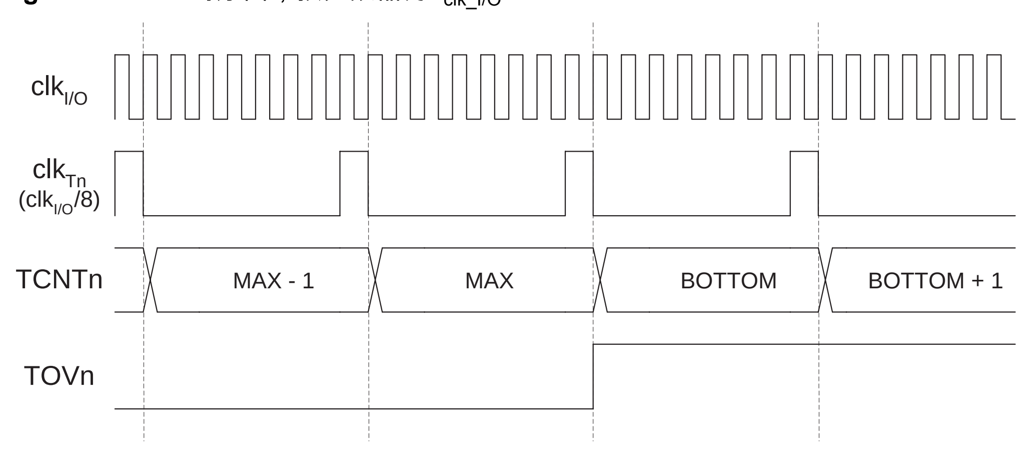


Figure 43 给出了各种模式下 (除了 CTC 模式)OCF0 的置位情况。

Figure 43. T/C 时序图， OCF0 置位，预分频器为 $f_{clk_I/O}/8$

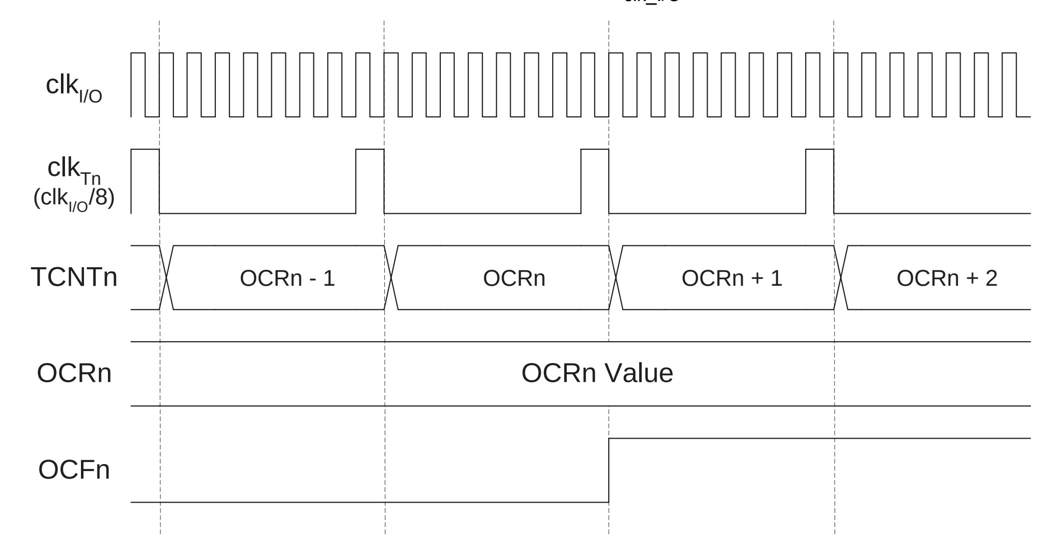
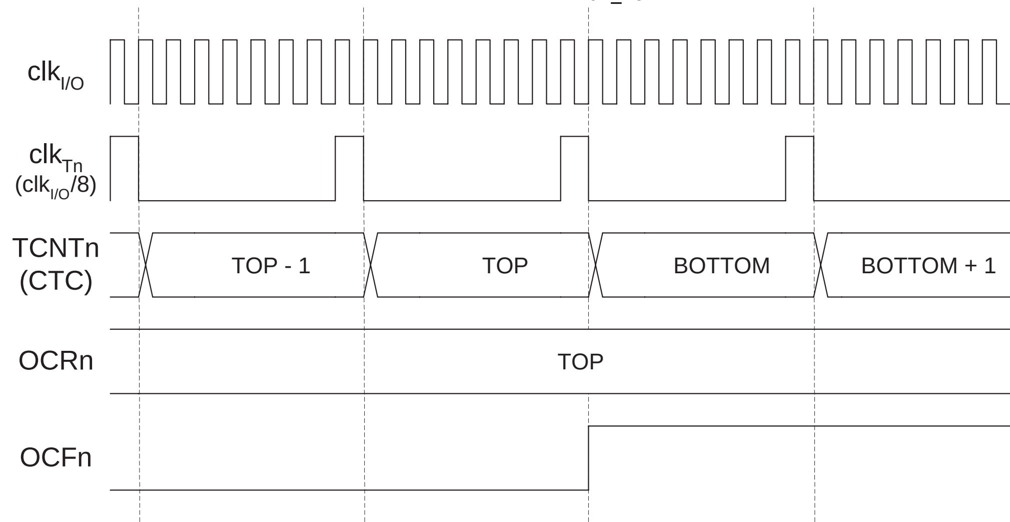


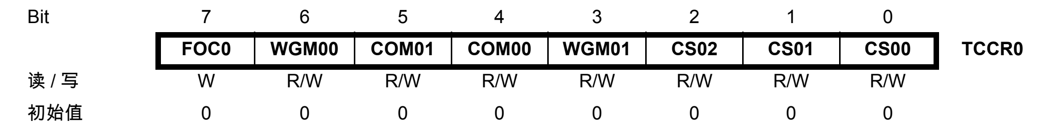
Figure 44 给出了 CTC 模式下 OCF0 置位和 TCNT0 清除的情况。

Figure 44. T/C 时序图， CTC 模式，预分频器为 $f_{clk_I/O}/8$



8 位 T/C 寄存器说明

T/C 控制寄存器 - TCCR0



Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
读 / 写	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC0: 强制输出比较**

FOC0 仅在 WGM 指明非 PWM 模式时才有效。但是，为了保证与未来器件的兼容性，在使用 PWM 时，写 TCCR0 要对其清零。对其写 1 后，波形发生器将立即进行比较操作。比较匹配输出引脚 OC0 将按照 COM01:0 的设置输出相应的电平。要注意 FOC0 仅仅是一个启动信号，真正对强制输出比较起作用的是 COM01:0 的设置。

FOC0 不会引发任何中断，也不会在使用 OCR0 作为 TOP 的 CTC 模式下对定时器进行清零。

读 FOC0 的返回值永远为 0。

- **Bit 6, 3 – WGM01:0: 波形产生模式**

这几位控制计数器的计数序列，计数器最大值 TOP 的来源，以及使用何种波形。T/C 支持的模式有：普通模式，比较匹配发生时清除计数器模式 (CTC)，以及两种 PWM 模式，如 Table 52 和 P 89“ 工作模式 ” 。

Table 52. 波形产生模式的位定义

模式	WGM01[(1)] (CTC0)	WGM00[(1)] (PWM0)	T/C 的工作模式	TOP	OCR0 的更新时间	TOV0 的置位时刻
0	0	0	普通	0xFF	立即更新	MAX
1	0	1	PWM，相位修正	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	立即更新	MAX
3	1	1	快速 PWM	0xFF	TOP	MAX

Note: 1. CTC0和PWM0已经不再使用了，要使用WGM01:0。但是功能和位置与以前版本兼容。

- **Bit 5:4 – COM01:0: 比较匹配输出模式**

这些位控制输出比较引脚 OC0 的行为。若 COM01:0 的任意一位或两位都置位，OC0 输出功能将重载普通端口功能。此时数据方向寄存器 (DDR) 需要按照 OC0 功能进行设置。

当 OC0 连接到物理引脚上时， COM01:0 的功能依赖于 WGM01:0 的设置。 Table 53 给出了当 WGM01:0 设置为普通模式或 CTC 模式时 COM01:0 的功能。

Table 53. 比较输出模式，非 PWM 模式

COM01	COM00	说明
0	0	正常的端口操作， OC0 未连接
0	1	比较匹配发生时 OC0 取反
1	0	比较匹配发生时 OC0 清零
1	1	比较匹配发生时 OC0 置位

Table 54 给出了当 WGM01:0 设置为快速 PWM 模式时 COM01:0 的功能。

Table 54. 比较输出模式，快速 PWM 模式 [(1)]

COM01	COM00	说明
0	0	正常的端口操作， OC0 未连接
0	1	保留
1	0	比较匹配发生时 OC0 清零，计数到 TOP 时 OC0 置位
1	1	比较匹配发生时 OC0 置位，计数到 TOP 时 OC0 清零

Note: 1. 一个特殊情况是 OCR0 等于 TOP，且 COM01 置位。此时比较匹配将被忽略，而计数到 TOP 时的动作继续有效。详细信息请参见 P 90“ 快速 PWM 模式 ” 。

Table 55 给出了当 WGM01:0 设置为相位修正 PWM 模式时 COM01:0 的功能。



Table 55. 比较输出模式，相位修正 PWM 模式 [1]

COM01	COM00	说明
0	0	正常的端口操作， OC0 未连接
0	1	保留
1	0	在升序计数时发生比较匹配将清零 OC0 ；降序计数时发生比较匹配将置位 OC0
1	1	在升序计数时发生比较匹配将置位 OC0 ；降序计数时发生比较匹配将清零 OC0

Note: 1. 一个特殊情况是 OCR0 等于 TOP ，且 COM01 置位。此时比较匹配将被忽略，而计数到 TOP 时的动作继续有效。详细信息请参见 P 92" 相位修正 PWM 模式 " 。

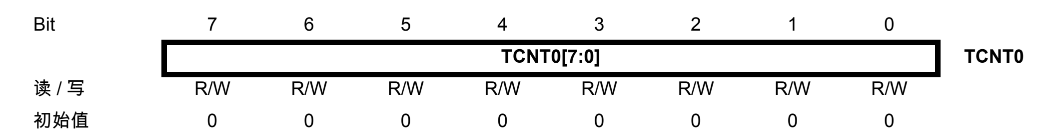
- **Bit 2:0 – CS02:0: 时钟选择**

用于选择 T/C 的时钟源。参见 Table 56。

Table 56. 时钟选择位定义

CS02	CS01	CS00	说明
0	0	0	无时钟， T/C 不工作
0	0	1	clk_{T0S}/(没有预分频)
0	1	0	clk_{T0S}/8 (来自预分频器)
0	1	1	clk_{T0S}/32 (来自预分频器)
1	0	0	clk_{T0S}/64 (来自预分频器)
1	0	1	clk_{T0S}/128 (来自预分频器)
1	1	0	clk_{T0S}/256 (来自预分频器)
1	1	1	clk_{T0S}/1024 (来自预分频器)

T/C 寄存器 - TCNT0



通过 T/C 寄存器可以直接对计数器的 8 位数据进行读写访问。对 TCNT0 寄存器的写访问将在下一个时钟阻止比较匹配。在计数器运行的过程中修改 TCNT0 的数值有可能丢失一次 TCNT0 和 OCR0 的比较匹配。

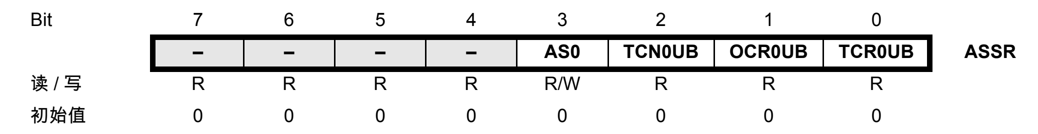
输出比较寄存器 - OCR0

Bit	7	6	5	4	3	2	1	0	
	OCR0[7:0]								OCR0
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器包含一个 8 位的数据，不间断地与计数器数值 TCNT0 进行比较。匹配事件可以用来产生输出比较中断，或者用来在 OC0 引脚上产生波形。

定时器 / 计数器的异步操作

异步状态寄存器 - ASSR



Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	AS0	TCN0UB	OCR0UB	TCR0UB	ASSR
读 / 写	R	R	R	R	R/W	R	R	R	
初始值	0	0	0	0	0	0	0	0	

- **Bit 3 – AS0: 异步 T/C0**

AS0为"0"时T/C0由I/O时钟$clk_{I/O}$驱动；AS0为"1"时T/C由连接到TOSC1引脚的晶体振荡器驱动。改变 AS0 有可能破坏 TCNT0、 OCR0 和 TCCR0 的内容。

- **Bit 2 – TCN0UB: T/C0 更新中**

T/C0工作于异步模式时，写TCNT0将引起TCN0UB置位。当TCNT0从暂存寄存器更新完毕后 TCN0UB 由硬件清零。 TCN20UB 为 0 表明 TCNT0 可以写入新值了。

- **Bit 1 – OCR0UB: 输出比较寄存器 0 更新中**

T/C0工作于异步模式时，写OCR0将引起OCR0UB置位。当OCR0从暂存寄存器更新完毕后 OCR0UB 由硬件清零。 OCR0UB 为 0 表明 OCR0 可以写入新值了。

- **Bit 0 – TCR0UB:T/C 控制寄存器 0 更新中**

T/C0工作于异步模式时，写TCCR0将引起TCR0UB置位。当TCCR0从暂存寄存器更新完毕后 TCR0UB 由硬件清零。 TCR0UB 为 0 表明 TCCR0 可以写入新值了。

如果在更新忙标志置位的时候写上述任何一个寄存器都将引起数据的破坏，并引发不必要的中断。

对 TCNT0，OCR0 和 TCCR0 进行读取的机制是不同的。读到的 TCNT0 为实际的值，而 OCR0 和 TCCR0 则是从暂存寄存器中读取的。

T/C0 的异步操作

T/C0 异步工作时要考虑如下几点：

- 警告：在同步和异步模式之间的转换有可能造成 TCNT0、OCR0、TCCR0 数据的损毁。安全的步骤应该是：
  1. 清零 OCIE0 和 TOIE0 以关闭 T/C0 的中断。
  2. 设置 AS0 以选择合适的时钟源。
  3. 位 TCNT0、 OCR0 和 TCCR0 写入新的数值。
  4. 等待 TCN0UB、 OCR0UB 和 TCR0UB 清零。
  5. 清除 T/C0 的中断标志。
  6. 需要的话使能中断。
- 振荡器对 32.768 kHz 的晶振进行了优化，从 TOSC1 输入外部时钟信号有可能导致不正确的 T/C0 操作。另外，系统主时钟必须比钟振高 4 倍以上。
- 写 TCNT0，OCR0 和 TCCR0 时数据首先送入暂存器，两个 TOSC1 正跳变后才锁存。在数据从暂存器写入目的寄存器之前不能写入新的数值。 3 个寄存器具有各自独立的暂存器，因此写 TCNT0 不会干扰写 OCR0。可以通过异步状态寄存器 ASSR 检查数据是否已经写入到目的寄存器。
- 如果要用 T/C0 作为 MCU 的唤醒条件，则在 TCNT0，OCR0 和 TCCR0 更新结束之前不能进入省电模式或扩展 Standby 模式，否则 MCU 可能会在 T/C0 设置生效之前进入休眠模式。这对于用 T/C0 的比较匹配中断唤醒 MCU 尤其重要，因为在更新 OCR0或TCNT0时比较匹配时禁止的。如果在更新完成之前(OCR0UB为0)MCU就进入了休眠模式，则比较匹配中断永远不会发生， MCU 也永远无法唤醒了。
- 如果要用T/C0作为省电模式或扩展Standby模式的唤醒条件，必须注意重新进入这些模式的过程。中断逻辑需要一个 TOSC1 周期进行复位。如果从唤醒到重新进入睡眠



的时间小于一个 TOSC1 周期，中断将不再发生，器件也无法唤醒。如果用户怀疑自己程序是否满足这一条件，可以采取如下方法：

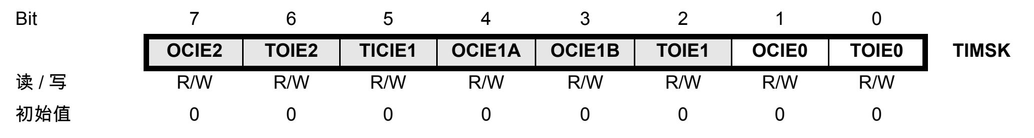
1. 对 TCCR0、 TCNT0 或 OCR0 写入合适的数据。
2. 等待 ASSR 相应的更新忙标志变低。
3. 进入省电模式或扩展 Standby 模式。

- 若选择了异步工作模式，T/C0 的 32.768 kHz 振荡器将一直工作，除非进入掉电模式或 Standby 模式。用户应该注意，此振荡器的稳定时间可能长达 1 秒钟。因此，建议用户在器件从睡眠模式唤醒或上电时至少等待 1 秒钟后再使用 T/C0。同时，由于启动过程时钟的不稳定性，唤醒时 T/C0 的数据不可使用。
- 省电模式或扩展 Standby 模式唤醒过程：中断条件满足后，在下一个定时器时钟唤醒过程启动。也就是说，至少一个定时器时钟后处理器才可以读到计数器的数值。唤醒后的 4 个时钟里 MCU 停止，接着执行中断例程。结束中断例程之后开始 SLEEP 之后的程序。
- 从省电模式唤醒之后的短时间内读取 TCNT0 可能返回不正确的数据。因为 TCNT0 是由异步的 TOSC 时钟驱动的，而读取 TCNT0 必须通过一个与内部 I/O 时钟同步的寄存器来完成。同步发生于每个 TOSC1 的上升沿。从省电模式唤醒后 I/O 时钟重新激活，而读到的 TCNT0 数值为进入睡眠模式前的值，直到下一个 TOSC1 上升沿的到来。从省电模式唤醒时的 TOSC1 相位是完全不可预测的。因此，读取 TCNT0 的推荐序列为：

1. 写一个任意数值到 OCR0 或 TCCR0 。
2. 等待相应的更新忙标志清零。
3. 读 TCNT0。

- 在异步模式下，中断标志的同步需要 3 个处理器周期加一个定时器周期。因此，至少一个定时器时钟后处理器才可以读到引起中断标志置位的计数器数值。输出比较引脚的变化与定时器时钟同步，而不是处理器时钟。

定时器 / 计数器中断屏蔽寄存器 - TIMSK



Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 1 – OCIE0: T/C0 输出比较匹配中断使能**

当 OCIE0 和状态寄存器的全局中断使能位 I 都为 '1' 时，T/C0 的输出比较匹配中断使能。当 T/C0 的比较匹配发生，即 TIFR 中的 OCF0 置位时，中断例程得以执行。

- **Bit 0 – TOIE0: T/C0 溢出中断使能**

当 OCIE0 和状态寄存器的全局中断使能位 I 都为 '1' 时， T/C0 的溢出中断使能。当 T/C0 发生溢出，即 TIFR 中的 TOV0 位置位时，中断例程得以执行。

定时器 / 计数器中断标志寄存器 - TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 1 – OCF0: 输出比较标志 0**

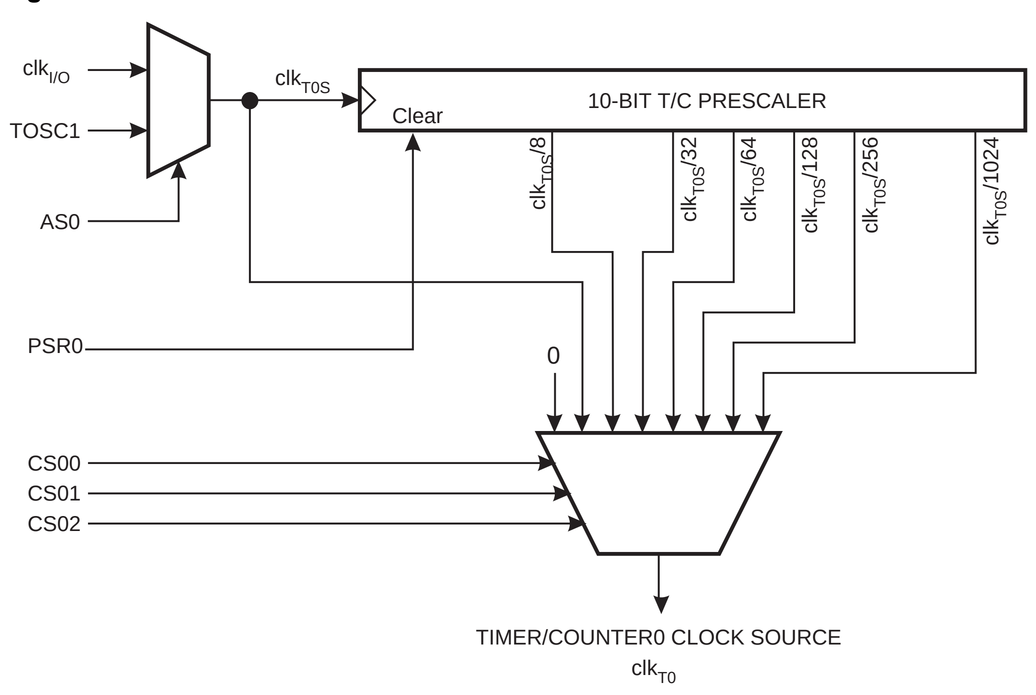
当 T/C0 与 OCR0(输出比较寄存器 0) 的值匹配时， OCF0 置位。此位在中断例程里硬件清零，或者通过对其写 1 来清零。当 SREG 中的位 I、 OCIE0 和 OCF0 都置位时，中断例程得到执行。

- **Bit 0 – TOV0:T/C0 溢出标志**

当 T/C0 溢出时， TOV0 置位。执行相应的中断例程时此位硬件清零。此外， TOV0 也可以通过写 1 来清零。当 SREG 中的位 I、TOIE0 和 TOV0 都置位时，中断例程得到执行。在 PWM 模式中，当 T/C0 在 $00 改变记数方向时， TOV0 置位。

定时器 / 计数器预分频器

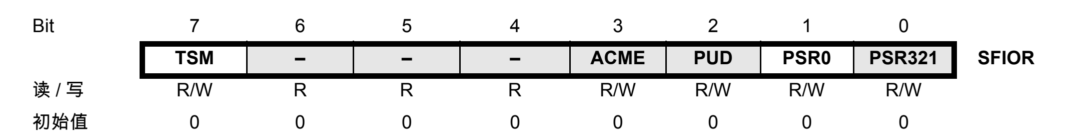
Figure 45. T/C0 的预分频器



T/C0预分频器的输入时钟称为clk_{T0}。缺省地，clk_{T0}与系统主时钟$clk_{I/O}$连接。若置位ASSR 的 AS0，T/C0 将由引脚 TOSC1 异步驱动，使得 T/C0 可以作为一个实时时钟。如果 AS0 置位，则 TOSC1 和 TOSC2 从端口 C 脱离。引脚上即可外接一个时钟晶振 (内部振荡器针对 32.768 kHz 的钟表晶体进行了优化)。不推荐在 TOSC1 上直接施加时钟信号。

T/C0 的可能预分频选项有：clk_{T0S}/8、 clk_{T0S}/32、 clk_{T0S}/64、 clk_{T0S}/128、 clk_{T0S}/256 和 clk_{T0S}/1024。此外还可以选择clk_{T0S}和0 (停止工作)。置位SFIOR寄存器的PSR0将复位预分频器，从而允许用户从可预测的预分频器开始工作。

特殊功能 IO 寄存器 - SFIOR



Bit	7	6	5	4	3	2	1	0	
	TSM	–	–	–	ACME	PUD	PSR0	PSR321	SFIOR
读 / 写	R/W	R	R	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – TSM:T/C 同步模式**

TSM置位时寄存器PSR0和PSR321保持其数据直到被更新，或者TSM被清零。此模式对同步 T/C 非常有用。通过设置 TSM 和合适的 PSR，相关的 T/C 将停止工作，然后被配置为具有相同的数值。一旦 TSM 清零，这些 T/C 立即同时开始计数。

- **Bit 1 – PSR0: T/C0 预分频器复位**

置位时 T/C0 的预分频器复位。操作完成后这一位由硬件自动清零。写入零时不会引发任何动作。若 T/C0 是由内部 CPU 时钟驱动的，则此位读返回值永远为零。如果 T/C0 工作于异步模式，则这一位置位后一直保持到预分频器复位操作真正完成。

16 位定时器 / 计数器 (定时器 / 计数器 1 和定时器 / 计数器 3)

16 位的 T/C 可以实现精确的程序定时 (事件管理)、波形产生和信号测量。其主要特点如下

- **真正的 16 位设计 (即允许 16 位的 PWM)**
- **3 个独立的输出比较单元**
- **双缓冲的输出比较寄存器**
- **一个输入比较单元**
- **输入捕捉噪声抑制器**
- **比较匹配发生时清除寄存器 (自动重载)**
- **无毛刺的相位修正 PWM**
- **可变的 PWM 周期**
- **频率发生器**
- **外部事件计数器**
- **10 个独立的中断源 (TOV1、 OCF1A、OCF1B、OCF1C、ICF1、TOV3、OCF3A、OCF3B、OCF3C 和 ICF3)**

ATmega103 兼容模式的限制

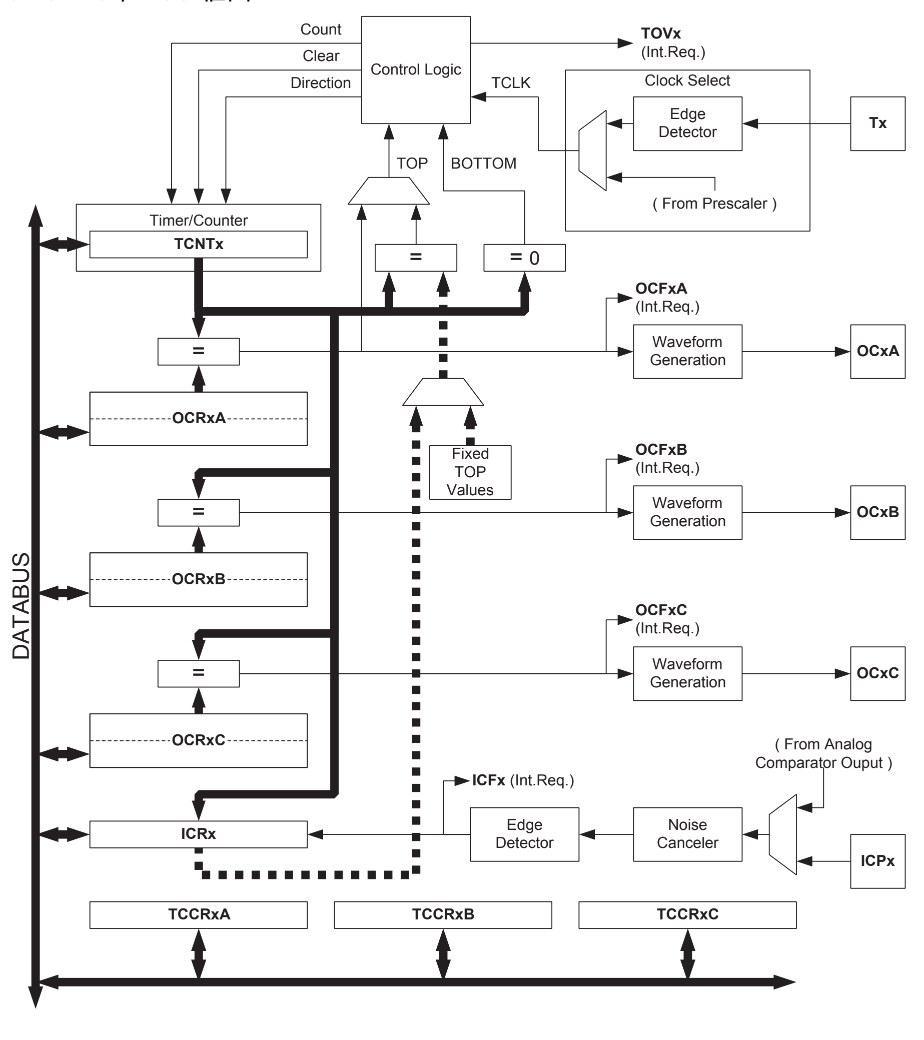
在 ATmega103 兼容模式下只有一个 16 位的 T/C(T/C1) , 而且只有两个比较寄存器 (比较 A 和比较 B)。

综述

本文多数的寄存器和位定义以通用的方式完成。 “n” 表示 T/C 通道号 , “x” 表示输出比较通道号。但是在写程序时要用完整的、精确的名称。

16 位 T/C 的简化框图示于 Figure 46。I/O 引脚位置见 P 2“引脚配置” 。CPU 可访问的 I/O 寄存器 , 包括 I/O 位和 I/O 引脚以粗体表示。物理 I/O 寄存器和位地址列于 P 120“16 位定时器 / 计数器寄存器说明 ” 。

Figure 46. 16 位 T/C 框图



Note: 请参考 P 2Figure 1 ， P 69Table 30 和 P 75Table 39 以获得 T/C1 和 T/C3 的引脚定义。

寄存器

定时器 / 计数器 TCNTn、输出比较寄存器 OCRnA/B/C 与输入捕捉寄存器 ICRn 均为 16 位寄存器。访问 16 位寄存器必须通过特殊的步骤，详见 P 102“ 访问 16 位寄存器 ” 。T/C 控制寄存器 TCCRnA/B/C 为 8 位寄存器，没有 CPU 访问的限制。中断请求 (图中简写为 Int.Req.) 信号在中断标志寄存器 TIFRn 与扩展定时中断标志寄存器 ETIFR 都有反映。所有中断都可以由中断屏蔽寄存器 TIMSKn 及扩展定时中断屏蔽寄存器 ETIMSK 控制。图中未给出 (E)TIFRn 与 (E)TIMSKn。

T/C 可由内部时钟通过预分频器或通过由 Tn 引脚输入的外部时钟驱动。引发 T/C 数值增加 (或减少) 的时钟源及其有效沿由时钟选择逻辑模块控制。没有选择时钟源时 T/C 处于停止状态。时钟选择逻辑模块的输出称为 clk_{Tn}。

双缓冲输出比较寄存器 OCRnA/B/C 一直与 T/C 的值做比较。波形发生器用比较结果产生 PWM 或在输出比较引脚 OCnA/B/C 输出可变频率的信号。参见 P 108 “输出比较单元” 。比较匹配结果还可置位比较匹配标志 OCFnA/B/C，用来产生输出比较中断请求。

当输入捕捉引脚 ICPn 或模拟比较器输入引脚 (见 P 210 “ 模拟比较器 ”) 有输入捕捉事件产生 (边沿触发) 时，当时的 T/C 值被传输到输入捕捉寄存器保存起来。输入捕捉单元包括一个数字滤波单元 (噪声消除器) 以降低噪声干扰。



在某些操作模式下， TOP 值或 T/C 的最大值可由 OCRnA 寄存器、 ICRn 寄存器，或一些固定数据来定义。在 PWM 模式下用 OCRnA 作为 TOP 值时， OCRnA 寄存器不能用作 PWM 输出。但此时 OCRnA 是双向缓冲的， TOP 值可在运行过程中得到改变。当需要一个固定的 TOP 值时可以使用 ICRn 寄存器，从而释放 OCRnA 来用作 PWM 的输出。

定义

以下定义适用于本节：

Table 57. 定义

BOTTOM	计数器计到 0x0000 时即达到 BOTTOM
MAX	计数器计到 0xFFFF (十进制的 65535) 时即达到 MAX
TOP	计数器计到计数序列的最大值时即达到 TOP。 TOP 值可以为固定值 0x00FF、0x01FF 或 0x03FF，或是存储于寄存器 OCRnA 或 ICRn 里的数值，具体有赖于工作模式

兼容性

16位T/C是从以前版本的16位AVRT/C改进和升级得来的。它在如下方面与以前的版本完全兼容：

- 包括定时器中断寄存器在内的所有 16 位 T/C 相关的 I/O 寄存器的地址。
- 包括定时器中断寄存器在内的所有 16 位 T/C 相关的寄存器位定位。
- 中断向量。

下列控制位名称已改，但具有相同的功能与寄存器单元：

- PWMn0 改为 WGMn0。
- PWMn1 改为 WGMn1。
- CTCn 改为 WGMn2。

16 位 T/C 控制寄存器中添加了下列寄存器：

- T/C 控制寄存器 C (TCCRnC)。
- 输出比较寄存器 C，含有 OCRnCH 与 OCRnCL。

16 位 T/C 控制寄存器中添加了下列位：

- TCCR1A 中加入 COM1C1:0。
- TCCRnC 中加入 FOCnA、 FOCnB 与 FOCnC。
- TCCRnB 中加入 WGMn3。

加入输出比较单元 C 的中断标志与屏蔽位。

16 位 T/C 的一些改进在某些特殊情况下将影响兼容性。

访问 16 位寄存器

TCNTn、OCRnA/B/C与ICRn是AVR CPU 通过8位数据总线可以访问的16位寄存器。读写 16 位寄存器需要两次操作。每个 16 位计时器都有一个 8 位临时寄存器用来存放其高 8 位数据。每个 16 位定时器所属的 16 位寄存器共用相同的临时寄存器。访问低字节会触发 16 位读或写操作。当 CPU 写入数据到 16 位寄存器的低字节时，写入的 8 位数据与存放在临时寄存器中的高 8 位数据组成一个 16 位数据，同步写入到 16 位寄存器中。当 CPU 读取 16 位寄存器的低字节时，高字节内容在读低字节操作的同时被放置于临时辅助寄存器中。

并非所有的 16 位访问都涉及临时寄存器。对 OCRnA/B/C 寄存器的读操作就不涉及临时寄存器。

写 16 位寄存器时，应先写入该寄存器的高位字节。而读 16 位寄存器时应先读取该寄存器的低位字节。

下面的例程说明了如何访问 16 位定时器寄存器。前提是假设不会发生更新临时寄存器内容的中断。同样的原则也适用于对 OCRnA/B/C 与 ICRn 寄存器的访问。使用 “C” 语言时，编译器会自动处理 16 位操作。

汇编代码例程 [1]

```
...
; 设置 TCNTn 为 0x01FF
ldi r17,0x01
ldi r16,0xFF
out TCNTnH,r17
out TCNTnL,r16
; 将 TCNTn 读入 r17:r16
in  r16,TCNTnL
in  r17,TCNTnH
...
```

C 代码例程 [1]

```
unsigned int i;
...
/* 设置 TCNTn 为 0x01FF */
TCNTn = 0x1FF;
/* 将 TCNTn 读入 i */
i = TCNTn;
...
```

Note: 1. 本代码假定已经包含了合适的头文件。当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、 “OUT”、 “SBIS”、“SBIC”、 “CBI” 与 “SBI” 指令。

汇编代码例程中 TCNTn 的返回值在 r17:r16 寄存器对中。

注意到 16 位寄存器的访问是一个基本操作是非常重要的。在对 16 位寄存器操作时，最好首先屏蔽中断响应，防止在主程序读写 16 位寄存器的两条指令之间发生这样的中断：它也访问同样的寄存器或其他的 16 位寄存器，从而更改了临时寄存器。如果这种情况发生，那么中断返回后临时寄存器中的内容已经改变，造成主程序对 16 位寄存器的读写错误。





下面的例程给出了读取 TCNTn 寄存器内容的基本操作。对 OCRnA/B/C 或 ICRn 的读操作可以使用相同的方法。

汇编代码例程 (1)

```
TIM16_ReadTCNTn:
  ; 保存全局中断标志
  in  r18,SREG
  ; 禁用中断
  cli
  ; 将 TCNTn 读入 r17:r16
  in  r16,TCNTnL
  in  r17,TCNTnH
  ; 恢复全局中断标志
  out SREG,r18
  ret
```

C 代码例程 (1)

```
unsigned int TIM16_ReadTCNTn( void )
{
  unsigned char sreg;
  unsigned int i;
  /* 保存全局中断标志 */
  sreg = SREG;
  /* 禁用中断 */
  _CLI();
  /* 将 TCNTn 读入 i */
  i = TCNTn;
  /* 恢复全局中断标志 */
  SREG = sreg;
  return i;
}
```

Note: 1. 本代码假定已经包含了合适的头文件。当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、 “OUT”、 “SBIS”、“SBIC”、 “CBI” 与 “SBI” 指令。

汇编代码例程中 TCNTn 的返回值在 r17:r16 寄存器对中。

下面的例程给出了写 TCNTn 寄存器的基本操作。对 OCRnA/B/C 或 ICRn 的写操作可以使用相同的方法。

汇编代码例程 [1]

```
TIM16_WriteTCNTn:
  ; 保存全局中断标志
  in  r18,SREG
  ; 禁用中断
  cli
  ; 设置 TCNTn 到 r17:r16
  out TCNTnH,r17
  out TCNTnL,r16
  ; 恢复全局中断标志
  out SREG,r18
  ret
```

C 代码例程 [1]

```
void TIM16_WriteTCNTn( unsigned int i )
{
  unsigned char sreg;
  unsigned int i;
  /* 保存全局中断标志 */
  sreg = SREG;
  /* 禁用中断 */
  _CLI();
  /* 设置 TCNTn 到 i */
  TCNTn = i;
  /* 恢复全局中断标志 */
  SREG = sreg;
}
```

Note: 1. 本代码假定已经包含了合适的头文件。当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、 “OUT”、 “SBIS”、“SBIC”、 “CBI” 与 “SBI” 指令。

汇编代码例程中 r17:r16 寄存器对保存的是 TCNTn 的写入数据。

重复利用暂存器的高字节

如果对不只一个 16 位寄存器写入数据而且所有的寄存器高字节相同，则只需写一次高字节。前面讲到的基本操作在这种情况下同样适用。



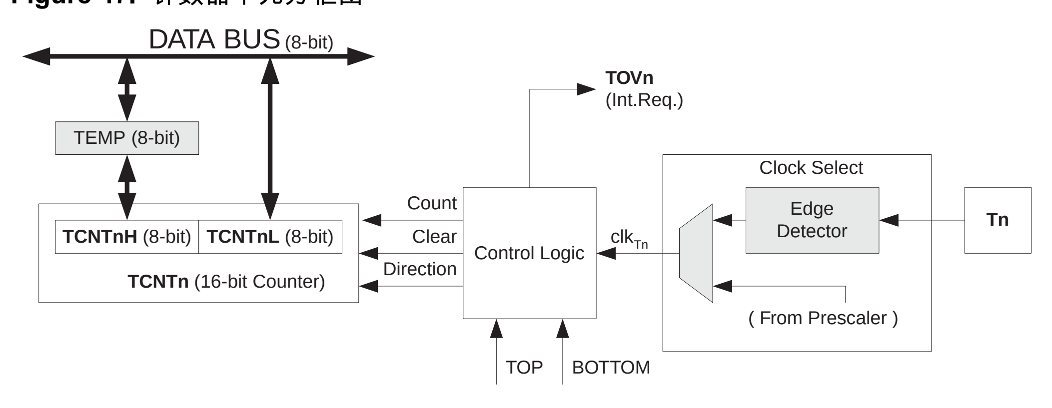
定时器 / 计数器的时钟源

T/C 时钟源可以来自内部，也可来自外部，由位于 T/C 控制寄存器 B(TCCRnB) 的时钟选择位 (CSn2:0) 决定。时钟源与预分频器的描述见 P 130“ 定时器 / 计数器 3、定时器 / 计数器 2 和定时器 / 计数器 1 的预分频器 ” 。

计数器单元

16 位 T/C 的主要部分是可编程的 16 位双向计数器单元。Figure 47 给出了计数器与其外围电路方框图。

Figure 47. 计数器单元方框图



信号描述 (内部信号) :

Count	TCNTn 加 1 或减 1。
Direction	确定是加操作还是减操作。
Clear	TCNTn 清零。
clk_{Tn}	定时器 / 计数器时钟信号。
TOP	表示 TCNTn 计数器到达最大值。
BOTTOM	表示 TCNTn 计数器到达最小值 (0)。

16 位计数器映射到两个 8 位 I/O 存储器位置:TCNTnH 为高 8 位，TCNTnL 为低 8 位。CPU 只能间接访问 TCNTnH 寄存器。CPU 访问 TCNTnH 时，实际访问的是临时寄存器 (TEMP)。读取TCNTnL时，临时寄存器的内容更新为TCNTnH的数值；而对TCNTnL执行写操作时，TCNTnH 被临时寄存器的内容所更新。这就使 CPU 可以在一个时钟周期里通过 8 位数据总线完成对 16 位计数器的读、写操作。此外还需要注意计数器在运行时的一些特殊情况。在这些特殊情况下对 TCNTn 写入数据会带来未知的结果。在合适的章节会对这些特殊情况进行具体描述。

根据工作模式的不同，在每一个 clk_{Tn} 时钟到来时，计数器进行清零、加 1 或减 1 操作。clk_{Tn} 由时钟选择位 CSn2:0 设定。当 CSn2:0= 0 时，计数器停止计数。不过 CPU 对 TCNTn 的读取与 clk_{Tn} 是否存在无关。 CPU 写操作比计数器清零和其他操作的优先级都高。

计数器的计数序列取决于寄存器 TCCRnA 和 TCCRnB 中标志位 WGMn3:0 的设置。计数器的运行 (计数) 方式与通过 OCnx 输出的波形发生方式有很紧密的关系。计数序列与波形产生的详细描述请参见 P 111“ 工作模式 ” 。

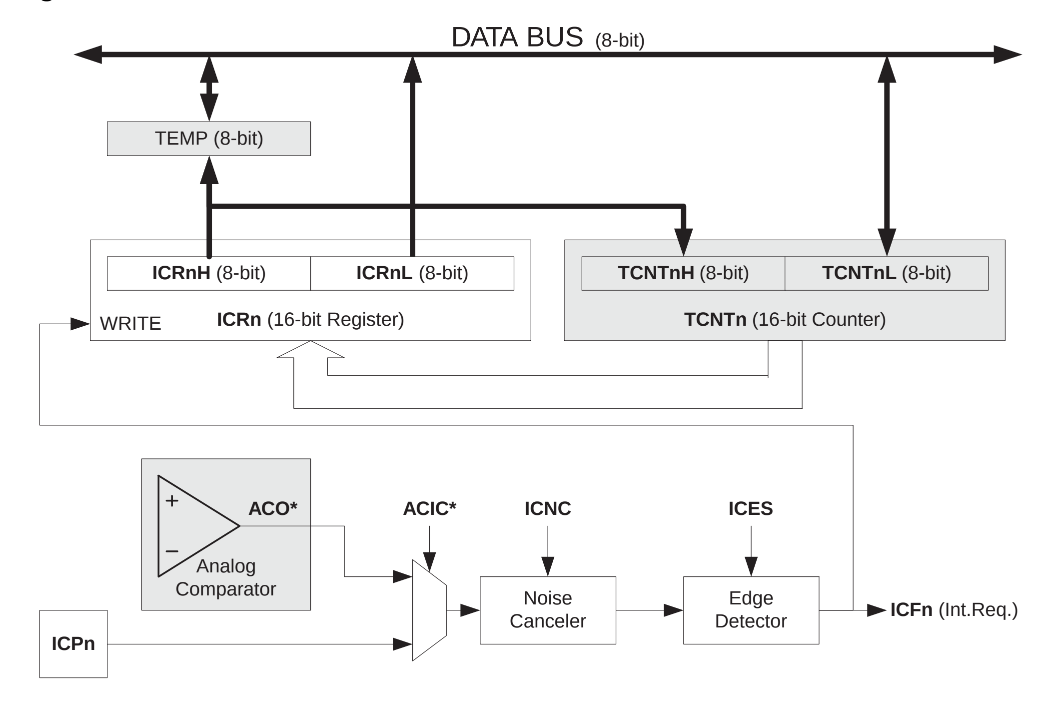
通过 WGMn3:0 确定了计数器的工作模式之后，TOVn 的置位方式也就确定了。TOVn 可以用来产生 CPU 中断。

输入捕捉单元

T/C 的输入捕捉单元可用来捕获外部事件，并为其赋予时间标记以说明此时间的发生时刻。外部事件发生的触发信号由引脚 ICPn 输入，也可通过模拟比较器单元来实现。时间标记可用来计算频率、占空比及信号的其它特征，以及为事件创建日志。

输入捕捉单元方框图见 Figure 48。图中不直接属于输入捕捉单元的部分用阴影表示。寄存器与位中的小写 “n” 表示定时器 / 计数器编号。

Figure 48. 输入捕捉单元方框图



Note: 模拟比较器输出 (ACO) 只可触发 T/C1 的 ICP – 而非 T/C3。

当引脚 ICPn 上的逻辑电平（事件）发生了变化，或模拟比较器输出 ACO 电平发生了变化，并且这个电平变化为边沿检测器所证实，输入捕捉即被激发：16 位的 TCNTn 数据被拷贝到输入捕捉寄存器 ICRn，同时输入捕捉标志位 ICFn 置位。如果此时 TICIEn = 1，输入捕捉标志将产生输入捕捉中断。中断执行时 ICFn 自动清零，或者也可通过软件在其对应的 I/O 位置写入逻辑 "1" 清零。

读取 ICRn 时要先读低字节 ICRnL，然后再读高字节 ICRnH。读低字节时，高字节被复制到高字节临时寄存器 TEMP。 CPU 读取 ICRnH 时将访问 TEMP 寄存器。

对 ICRn 寄存器的写访问只存在于波形产生模式。此时 ICRn 被用作计数器的 TOP 值。写 ICRn 之前首先要设置 WGMn3:0 以允许这个操作。对 ICRn 寄存器进行写操作时必须先将高字节写入 ICRnH I/O 位置，然后再将低字节写入 ICRnL。

请参见 P 102“ 访问 16 位寄存器 ” 以了解更多的关于如何访问 16 位寄存器的信息。

输入捕捉触发源

输入捕捉单元的主要触发源是 ICPn。T/C1 还可用模拟比较输出作为输入捕捉单元的触发源。用户必须通过设置模拟比较控制与状态寄存器 ACSR 的模拟比较输入捕捉位 ACIC 来做到这一点。要注意的是，改变触发源有可能造成一次输入捕捉。因此在改变触发源后必须对输入捕捉标志执行一次清零操作以避免出现错误的结果。

ICPn与ACO的采样方式与Tn引脚是相同的(P 130Figure 59),使用的边沿检测器也一样。但是使能噪声抑制器后，在边沿检测器前会加入额外的逻辑电路并引入 4 个系统时钟周期的延迟。要注意的是，除去使用 ICRn 定义 TOP 的波形产生模式外， T/C 中的噪声抑制器与边沿检测器总是使能的。

输入捕捉也可以通过软件控制引脚 ICPn 的方式来触发。



噪声抑制器

噪声抑制器通过一个简单的数字滤波方案提高系统抗噪性。它对输入触发信号进行 4 次采样。只有当 4 次采样值相等时其输出才会送入边沿检测器。

置位 TCCRnB 的 ICNCn 将使能噪声抑制器。使能噪声抑制器后 , 在输入发生变化到 ICRn 得到更新之间将会有额外的 4 个系统时钟周期的延时。噪声抑制器使用的是系统时钟 , 因而不受预分频器的影响。

使用输入捕捉单元

使用输入捕捉单元的最大问题就是分配足够的处理器资源来处理输入事件。事件的时间间隔是关键。如果处理器在下一次事件出现之前没有读取 ICRn 的数据， ICRn 就会被新值覆盖 , 从而无法得到正确的捕捉结果。

使用输入捕捉中断时 , 中断程序应尽可能早的读取 ICRn 寄存器。尽管输入捕捉中断优先级相对较高 , 但最大中断响应时间与其它正在运行的中断程序所需的时间相关。

在任何输入捕捉工作模式下都不推荐在操作过程中改变 TOP 值。

测量外部信号的占空比时要求每次捕捉后都要改变触发沿。因此读取 ICRn 后必须尽快改变敏感的信号边沿。改变边沿后 , ICFn 必须由软件清零 (在对应的 I/O 位置写 "1”)。若仅需测量频率 , 且使用了中断发生 , 则不需对 ICFn 进行软件清零。

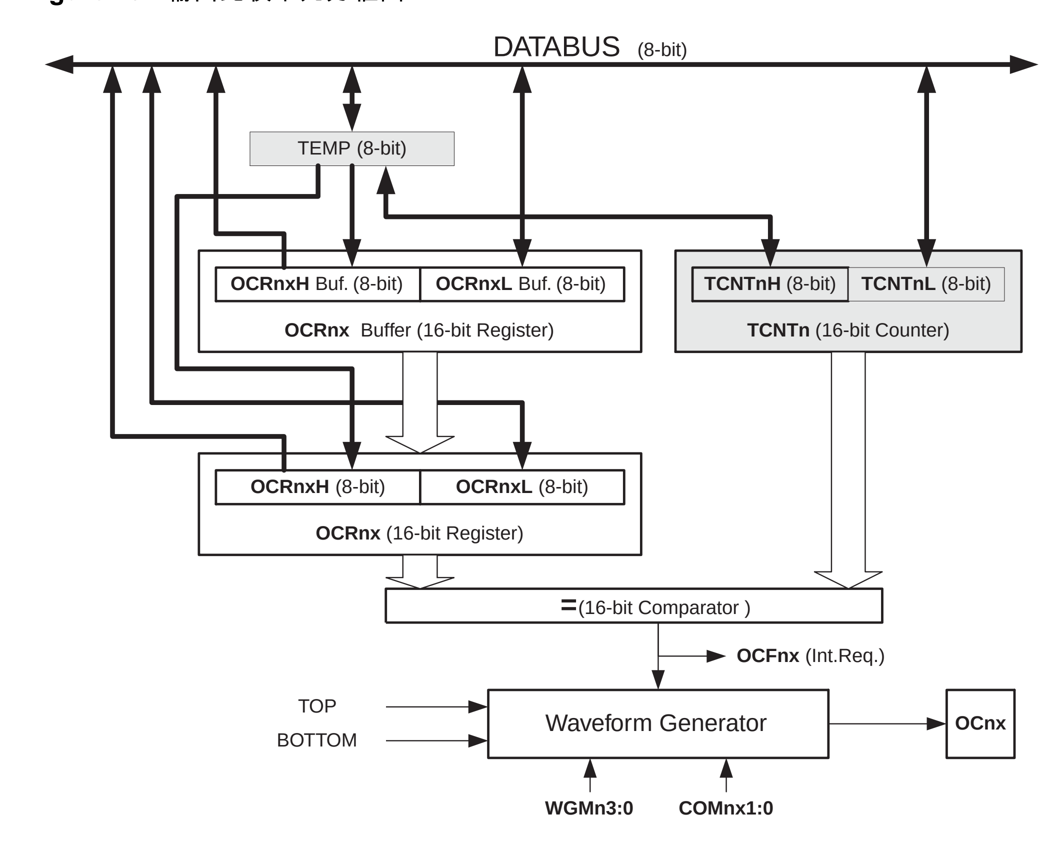
输出比较单元

16 位比较器持续比较 TCNTn 与 OCRnx 的内容 , 一旦发现它们相等 , 比较器立即产生一个匹配信号。然后 OCFnx 在下一个定时器时钟置位。如果此时 OCIEnx = 1 ， OCFnx 置位将引发输出比较中断。中断执行时 OCFnx 标志自动清零 , 或者通过软件在其相应的 I/O 位置写入逻辑 "1” 也可以清零。根据 WGMn3:0 与 COMnx1:0 的不同设置 , 波形发生器用匹配信号生成不同的波形。波形发生器利用 TOP 和 BOTTOM 信号处理在某些模式下对极值的操作 (P 111 “ 工作模式 ”)。

输出比较单元 A 的一个特质是定义 T/C 的 TOP 值 (即计数器的分辨率)。此外， TOP 值还用来定义通过波形发生器产生的波形的周期。

Figure 49 给出输出比较单元的方框图。寄存器与位上的小写 “n” 表示器件编号 (n = n 表示 T/Cn)， “x” 表示输出比较单元 (A/B/C)。框图中非输出比较单元部分用阴影表示。

Figure 49. 输出比较单元方框图



当 T/C 工作在 12 种 PWM 模式种的任意一种时， OCRnx 寄存器为双缓冲寄存器；而在正常工作模式和匹配时清零模式 (CTC) 双缓冲功能是禁止的。双缓冲可以实现 OCRnx 寄存器对 TOP 或 BOTTOM 的同步更新，防止产生不对称的 PWM 波形，消除毛刺。

访问 OCRnx 寄存器看起来很复杂，其实不然。使能双缓冲功能时，CPU 访问的是 OCRnx 缓冲寄存器；禁止双缓冲功能时 CPU 访问的则是 OCRnx 本身。 OCRnx(缓冲或比较) 寄存器的内容只有写操作才能将其改变 (T/C 不会自动将此寄存器更新为 TCNT1 或 ICR1 的内容)，所以 OCR1x 不用通过 TEMP 读取。但是象其他 16 位寄存器一样首先读取低字节是一个好习惯。由于比较是连续进行的，因此在写 OCR1x 时必须通过 TEMP 寄存器来实现。首先需要写入的是高字节 OCRnxH。当 CPU 将数据写入高字节的 I/O 地址时，TEMP 寄存器的内容即得到更新。接下来写低字节 OCRnxL。在此同时，位于 TEMP 寄存器的高字节数据被拷贝到 OCRnx 缓冲器，或是 OCRnx 比较寄存器。

请参见 P 102“ 访问 16 位寄存器 ” 以了解更多的关于如何访问 16 位寄存器的信息。

强制输出比较

工作于非 PWM 模式时，可以通过对强制输出比较位 FOCnx 写 ”1” 的方式来产生比较匹配。强制比较匹配不会置位 OCFnx 标志，也不会重载 / 清零定时器，但是 OCnx 引脚将被更新，好象真的发生了比较匹配一样 (COMx1:0 决定 OCnx 是置位、清零，还是交替变化)。

写 TCNTn 将阻止比较匹配

CPU对TCNTn寄存器的写操作会阻止比较匹配的发生。这个特性可以用来将OCRnx初始化为与 TCNTn 相同的数值而不触发中断。

使用输出比较匹配单元

由于在任意模式下写 TCNTn 都将在下一个定时器时钟周期里阻止比较匹配，在使用输出比较时改变TCNTn就会有风险，不管T/C是否在运行。若写入TCNTn的数值等于OCRnx，比较匹配就被忽略了，造成不正确的波形发生结果。在 PWM 模式下，当 TOP 为可变数

值时，不要赋予 TCNTn 和 TOP 相等的数值。否则会丢失一次比较匹配，计数器也将计到0xFFFF。类似地，在计数器进行降序计数时不要对TCNTn写入等于BOTTOM的数据。

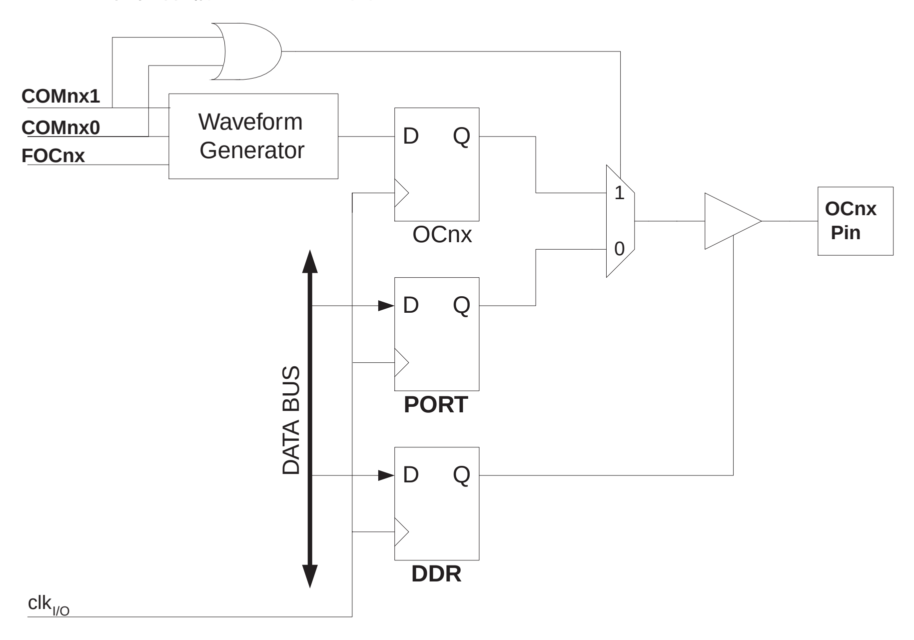
OCnx 的设置应该在设置数据方向寄存器之前完成。最简单的设置 OCnx 的方法是在普通模式下利用强制输出比较 FOCnx。即使在改变波形发生模式时 OCnx 寄存器也会一直保持它的数值。

COMnx1:0 和比较数据都不是双缓冲的。 COMnx1:0 的改变将立即生效。

比较匹配输出单元

比较匹配模式控制位 COMnx1:0 具有双重功能。波形发生器利用 COMnx1:0 来确定下一次比较匹配发生时的输出比较 OCnx 状态； COMnx1:0 还控制 OCnx 引脚输出的来源。Figure 50 为受 COMnx1:0 设置影响的逻辑的简化原理图。I/O 寄存器、I/O 位和 I/O 引脚以粗体表示。图中只给出了受 COMnx1:0 影响的通用 I/O 端口控制寄存器 (DDR 和 PORT)。谈及 OCnx 状态时指的是内部 OCnx 寄存器，而不是 OCnx 引脚的状态。系统复位时 COMnx 寄存器复位为 "0"。

Figure 50. 比较匹配输出单元原理图



只要 COMnx1:0 不全为零，波形发生器的输出比较功能就会重载 OCnx 的通用 I/O 口功能。但是 OCnx 引脚的方向仍旧受控于数据方向寄存器 (DDR)。从 OCnx 引脚输出有效信号之前必须通过数据方向寄存器的 DDR_OCnx 将此引脚设置为输出。一般情况下功能重载与波形发生器的工作模式无关，但也由一些例外，详见 Table 58、Table 59 与 Table 60。

输出比较逻辑的设计允许 OCnx 在输出之前首先进行初始化。要注意某些 COMnx1:0 设置在某些特定的工作模式下是保留的，如 P 120 “16 位定时器 / 计数器寄存器说明 ” 所示。

COMnx1:0 不影响输入捕捉单元。

比较匹配模式和波形产生

波形发生器利用 COMnx1:0 的方法在普通模式、 CTC 模式和 PWM 模式下有所区别。对于所有的模式，设置 COMnx1:0 = 0 表明比较匹配发生时波形发生器不会操作 OCnx 寄存器。非PWM模式的比较输出请参见P 120Table 58 ；快速PWM的比较输出于 P 121Table 59 ；相位修正及相频修正 PWM 的比较输出于 P 121Table 60 。

改变 COMnx1:0 将影响写入数据后的第一次比较匹配。对于非 PWM 模式，可以通过使用 FOCnx 来立即产生效果。

工作模式

工作模式 - T/C 和输出比较引脚的行为 - 由波形发生模式 (WGMn3:0) 及比较输出模式 (COMnx1:0) 的控制位决定。比较输出模式对计数序列没有影响，而波形产生模式对计数序列则有影响。 COMnx1:0 控制 PWM 输出是否为反极性。非 PWM 模式时 COMnx1:0 控制输出是否应该在比较匹配发生时置位、清零，或是电平取反 (P 111 “ 比较匹配输出单元 ”)。

具体的时序信息请参考 P 117“ 定时器 / 计数器时序图 ” 。



普通模式

普通模式 (WGMn3:0 = 0) 为最简单的工作模式。在此模式下计数器不停地累加。计到最大值后 (MAX = 0xFFFF) 由于数值溢出计数器简单地返回到最小值 0x0000 重新开始。在TCNTn为零的同一个定时器时钟里T/C溢出标志TOVn置位。此时TOVn有点象第17位，只是只能置位，不会清零。但由于定时器中断服务程序能够自动清零 TOVn，因此可以通过软件提高定时器的分辨率。在普通模式下没有什么需要特殊考虑的，用户可以随时写入新的计数器数值。

在普通模式下输入捕捉单元很容易使用。要注意的是外部事件的最大时间间隔不能超过计数器的分辨率。如果事件间隔太长，必须使用定时器溢出中断或预分频器来扩展输入捕捉单元的分辨率。

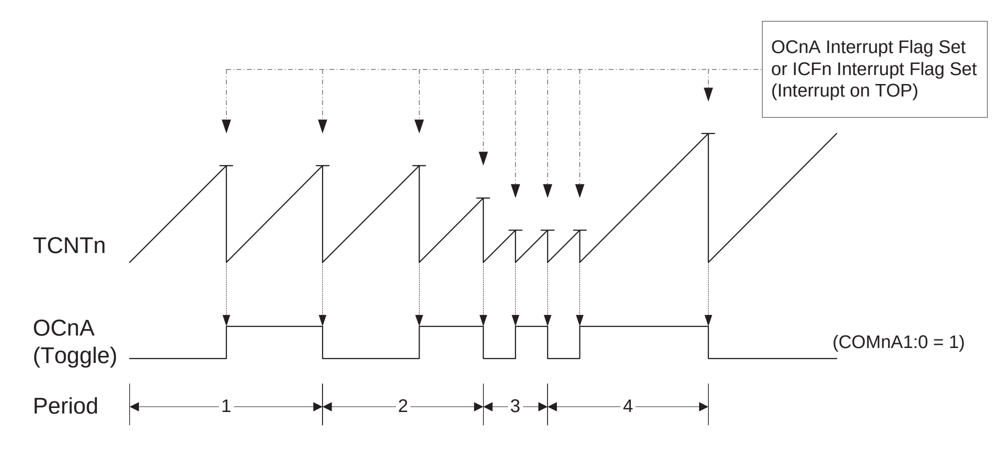
输出比较单元可以用来产生中断。但是不推荐在普通模式下利用输出比较来产生波形，因为会占用太多的 CPU 时间。

CTC(比较匹配时清除定时器) 模式

在 CTC 模式 (WGMn3:0 = 4 或 12) 里 OCRnA 或 ICRn 寄存器用于调节计数器的分辨率。当计数器的数值 TCNTn 等于 OCRnA(WGMn3:0 = 4) 或等于 ICRn (WGMn3:0 = 12) 时计数器清零。 OCRnA 或 ICRn 定义了计数器的 TOP 值，亦即计数器的分辨率。这个模式使得用户可以很容易地控制比较匹配输出的频率，也简化了外部事件计数的操作。

CTC模式的时序图为Figure 51。计数器数值TCNTn一直累加到TCNTn与OCRnA 或ICRn 匹配，然后 TCNTn 清零。

Figure 51. CTC 模式的时序图



利用 OCFnA 或 ICFn 标志可以在计数器数值达到 TOP 时产生中断。在中断服务程序里可以更新 TOP 的数值。由于 CTC 模式没有双缓冲功能，在计数器以无预分频器或很低的预分频器工作的时候将 TOP 更改为接近 BOTTOM 的数值时要小心。如果写入的 OCRnA 或 ICRn 的数值小于当前 TCNTn 的数值，计数器将丢失一次比较匹配。在下一次比较匹配发生之前，计数器不得不先计数到最大值 0xFFFF，然后再从 0x0000 开始计数到 OCRnA 或 ICRn。在许多情况下，这一特性并非我们所希望的。替代的方法是使用快速 PWM 模式，该模式使用 OCRnA 定义 TOP 值 (WGMn3:0 = 15)，因为此时 OCRnA 为双缓冲。

为了在CTC模式下得到波形输出，可以设置OCnA在每次比较匹配发生时改变逻辑电平。这可以通过设置 COMnA1:0 = 1 来完成。在期望获得 OCnA 输出之前，首先要将其端口设置为输出(DDR_OCnA = 1)。波形发生器能够产生的最大频率为$f_{OC0} = f_{clk_I/O}/2$ (OCRnA = 0x0000)。频率由如下公式确定：

$$f_{OCnA} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$

变量 N 代表预分频因子 (1、 8、 64、 256 或 1024)。

在普通模式下，TOVn 标志的置位发生在计数器从 MAX 变为 0x0000 的定时器时钟周期。

快速 PWM 模式

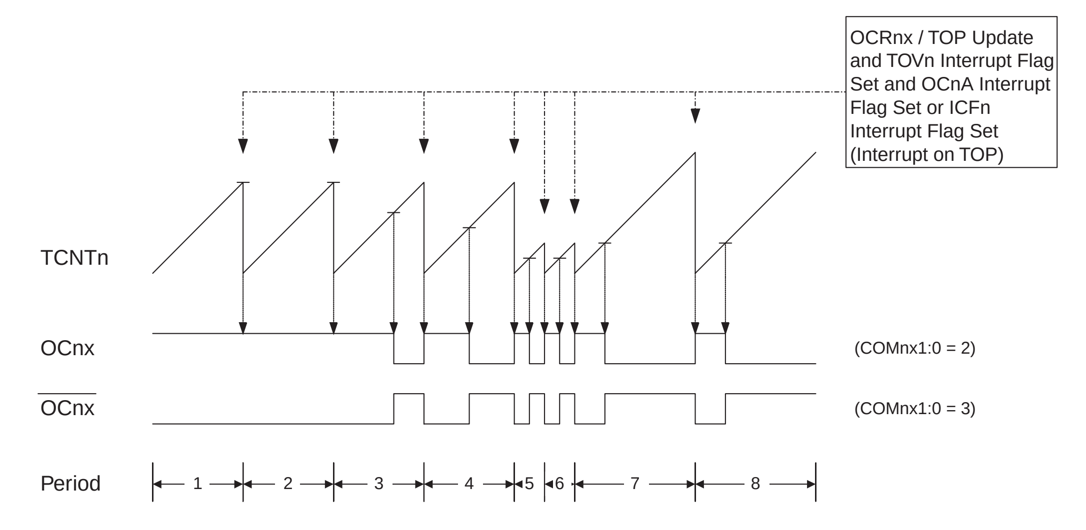
快速 PWM 模式 (WGMn3:0 = 5、 6、7、 14 或 15) 可用来产生高频的 PWM 波形。快速 PWM 模式与其他 PWM 模式的不同之处是其单边斜坡工作方式。计数器从 BOTTOM 计到 TOP，然后立即回到 BOTTOM 重新开始。对于普通的比较输出模式，输出比较引脚 OCnx 在 TCNTn 与 OCRnx 匹配时置位，在 TOP 时清零；对于反向比较输出模式， OCRnx 的动作正好相反。由于使用了单边斜坡模式，快速 PWM 模式的工作频率比使用双斜坡的相位修正 PWM 模式高一倍。此高频操作特性使得快速 PWM 模式十分适合于功率调节，整流和 DAC 应用。高频可以减小外部元器件 (电感，电容) 的物理尺寸，从而降低系统成本。

工作于快速 PWM 模式时，PWM 分辨率可固定为 8、9 或 10 位，也可由 ICRn 或 OCRnA 定义。最小分辨率为 2 比特 (ICRn 或 OCRnA 设为 0x0003)，最大分辨率为 16 位 (ICRn 或 OCRnA 设为 MAX)。 PWM 分辨率位数可用下式计算：

$$R_{FPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

工作于快速 PWM 模式时，计数器的数值一直累加到固定数值 0x00FF、0x01FF、0x03FF (WGMn3:0 = 5、6 或 7)、ICRn (WGMn3:0 = 14) 或 OCRnA (WGMn3:0 = 15)，然后在后面的一个时钟周期清零。具体的时序图为 Figure 52。图中给出了当使用 OCRnA 或 ICRn 来定义 TOP 值时的快速 PWM 模式。图中柱状的 TCNTn 表示这是单边斜坡操作。方框图同时包含了普通的 PWM 输出以及反向 PWM 输出。 TCNTn 斜坡上的短水平线表示 OCRnx 和 TCNTn 的匹配比较。比较匹配后 OCnx 中断标志置位。

Figure 52. 快速 PWM 模式时序图



计时器数值达到 TOP 时 T/C 溢出标志 TOVn 置位。另外若 TOP 值是由 OCRnA 或 ICRn 定义的，则 OCnA 或 ICFn 标志将与 TOVn 在同一个时钟周期置位。如果中断使能，可以在中断服务程序里来更新 TOP 以及比较数据。

改变TOP值时必须保证新的TOP值不小于所有比较寄存器的数值。否则TCNTn与OCRnx 不会出现比较匹配。使用固定的 TOP 值时，向任意 OCRnx 寄存器写入数据时未使用的位将屏蔽为 "0”。

定义 TOP 值时更新 ICRn 与 OCRnA 的步骤时不同的。ICRn 寄存器不是双缓冲寄存器。这意味着当计数器以无预分频器或很低的预分频工作的时候，给 ICRn 赋予一个小的数值时存在着新写入的 ICRn 数值比 TCNTn 当前值小的危险。结果是计数器将丢失一次比较匹配。在下一次比较匹配发生之前，计数器不得不先计数到最大值 0xFFFF，然后再从 0x0000 开始计数，直到比较匹配出现。而 OCRnA 寄存器则是双缓冲寄存器。这一特性决定 OCRnA 可以随时写入。写入的数据被放入 OCRnA 缓冲寄存器。在 TCNTn 与 TOP 匹配后的下一个时钟周期，OCRnA 比较寄存器的内容被缓冲寄存器的数据所更新。在同一个时钟周期 TCNTn 被清零，而 TOVn 标志被设置。



使用固定 TOP 值时最好使用 ICRn 寄存器定义 TOP。这样 OCRnA 就可以用于在 OCnA 输出 PWM 波。但是，如果 PWM 基频不断变化 (通过改变 TOP 值)， OCRnA 的双缓冲特性使其更适合于这个应用。

工作于快速 PWM 模式时，比较单元可以在 OCnx 引脚上输出 PWM 波形。设置 COMnx1:0 为 2 可以产生普通的 PWM 信号；为 3 则可以产生反向 PWM 波形 (参见 P 121Table 59)。此外，要真正从物理引脚上输出信号还必须将 OCnx 的数据方向 DDR_OCnx 设置为输出。产生 PWM 波形的机理是 OCnx 寄存器在 OCRnx 与 TCNTn 匹配时置位 (或清零)，以及在计数器清零 (从 TOP 变为 BOTTOM) 的那一个定时器时钟周期清零 (或置位)。

输出的 PWM 频率可以通过如下公式计算得到：

$$f_{OCnxPWM} = \frac{f_{\text{clk_I/O}}}{N \cdot (1 + TOP)}$$

变量 N 代表分频因子 (1、8、64、 256 或 1024)。

OCRnx 寄存器为极限值时说明了快速 PWM 模式的一些特殊情况。若 OCRnx 等于 BOTTOM(0x0000)，输出为出现在第TOP+1个定时器时钟周期的窄脉冲；OCRnx为TOP 时，根据 COMnx1:0 的设定，输出恒为高电平或低电平。

通过设定 OCnA 在比较匹配时进行逻辑电平取反 (COMnA1:0 = 1)，可以得到占空比为 50% 的周期信号。这只适用于 OCR1A 用来定义 TOP 值的情况 (WGM13:0 = 15)。OCRnA 为 0(0x0000) 时信号有最高频率 $f_{oc0} = f_{clk_I/O}/2$。这个特性类似于 CTC 模式下的 OCnA 取反操作，不同之处在于快速 PWM 模式具有双缓冲。

相位修正 PWM 模式

相位修正 PWM 模式 (WGMn3:0 = 1、2、3、10 或 11) 为用户提供了一个获得高精度的、相位准确的 PWM 波形的方法。与相位和频率修正模式类似，此模式基于双斜坡操作。计时器重复地从 BOTTOM 计到 TOP，然后又从 TOP 倒退回到 BOTTOM。在一般的比较输出模式下，当计时器往 TOP 计数时若 TCNTn 与 OCRnx 匹配，OCnx 将清零为低电平；而在计时器往 BOTTOM 计数时若 TCNTn 与 OCRnx 匹配， OCnx 将置位为高电平。工作于反向比较输出时则正好相反。与单斜坡操作相比，双斜坡操作可获得的最大频率要小。但其对称特性十分适合于电机控制。

相位修正 PWM 模式的 PWM 分辨率固定为 8、 9 或 10 位，或由 ICRn 或 OCRnA 定义。最小分辨率为 2 比特 (ICRn 或 OCRnA 设为 0x0003)，最大分辨率为 16 位 (ICRn 或 OCRnA 设为 MAX)。 PWM 分辨率位数可用下式计算：

$$R_{PCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

工作于相位修正 PWM 模式时，计数器的数值一直累加到固定值 0x00FF、 0x01FF、 0x03FF (WGMn3:0 = 1、2 或 3)、ICRn (WGMn3:0 = 10) 或 OCRnA (WGMn3:0 = 11)，然后改变计数方向。在一个定时器时钟里 TCNTn 值等于 TOP 值。具体的时序图为 Figure 53。图中给出了当使用 OCRnA 或 ICRn 来定义 TOP 值时的相位修正 PWM 模式。图中柱状的 TCNTn 表示这是双边斜坡操作。方框图同时包含了普通的 PWM 输出以及反向 PWM 输出。TCNTn 斜坡上的短水平线表示 OCRnx 和 TCNTn 的匹配比较。比较匹配后 OCnx 中断标志置位。

Figure 53. 相位修正 PWM 模式的时序图

计时器数值达到 BOTTOM 时 T/C 溢出标志 TOVn 置位。若 TOP 由 OCRnA 或 ICRn 定义，在 OCRnx 寄存器通过双缓冲方式得到更新的同一个时钟周期里 OCnA 或 ICFn 标志置位。标志置位后即可产生中断。

改变TOP值时必须保证新的TOP值不小于所有比较寄存器的数值。否则TCNTn与OCRnx不会出现比较匹配。使用固定的 TOP 值时，向任意 OCRnx 寄存器写入数据时未使用的位将屏蔽为 "0"。在 Figure 53 给出的第三个周期中，在 T/C 运行于相位修正模式时改变 TOP值导致了不对称输出。其原因在于OCRnx寄存器的更新时间。由于OCRnx的更新时刻为定时器 / 计数器达到 TOP 之时，因此 PWM 的循环周期起始于此，也终止于此。就是说，下降斜坡的长度取决于上一个 TOP 值，而上升斜坡的长度取决于新的 TOP 值。若这两个值不同，一个周期内两个斜坡长度不同，输出也就不对称了。

若要在 T/C 运行时改变 TOP 值，最好用相位与频率修正模式代替相位修正模式。若 TOP 保持不变，那么这两种工作模式实际没有区别。

工作于相位修正 PWM 模式时，比较单元可以在 OCnx 引脚输出 PWM 波形。设置 COMnx1:0 为 2 可以产生普通的 PWM，设置 COMnx1:0 为 3 可以产生反向 PWM (参见 P 121Table 60)。要真正从物理引脚上输出信号还必须将OCnx的数据方向DDR_OCnx设置为输出。OCRnx 和 TCNTn 比较匹配发生时 OCnx 寄存器将产生相应的清零或置位操作，从而产生 PWM 波形。工作于相位修正模式时 PWM 频率可由如下公式获得：

$$f_{OCnxPCPWM} = \frac{f_{clk_I/O}}{2 \cdot N \cdot TOP}$$

变量 N 表示预分频因子 (1、8、64、256 或 1024)。

OCRnx 寄存器处于极值时表明了相位修正 PWM 模式的一些特殊情况。在普通 PWM 模式下，若 OCRnx 等于 BOTTOM，输出一直保持为低电平；若 OCRnx 等于 TOP，输出则保持为高电平。反向 PWM 模式正好相反。

若 OCnA 作为 TOP 值 (WGMn3:0 = 11) 且 COMnA1:0 = 1，OCnA 输出占空比为 50%。

相位和频率修正 PWM 模式

相位与频率修正 PWM 模式 (WGMn3:0 = 8 或 9) - 以下简称相频修正 PWM 模式 - 可以产生高精度的、相位与频率都准确的 PWM 波形。与相位修正模式类似，相频修正 PWM 模

式基于双斜坡操作。计时器重复地从 BOTTOM 计到 TOP，然后又从 TOP 倒退回到 BOTTOM。在一般的比较输出模式下，当计时器往TOP计数时若TCNTn与OCRnx匹配，OCnx将清零为低电平；而在计时器往BOTTOM计数时TCNTn与OCRnx匹配，OCnx将置位为高电平。工作于反向输出比较时则正好相反。与单斜坡操作相比，双斜坡操作可获得的最大频率要小。但其对称特性十分适合于电机控制。

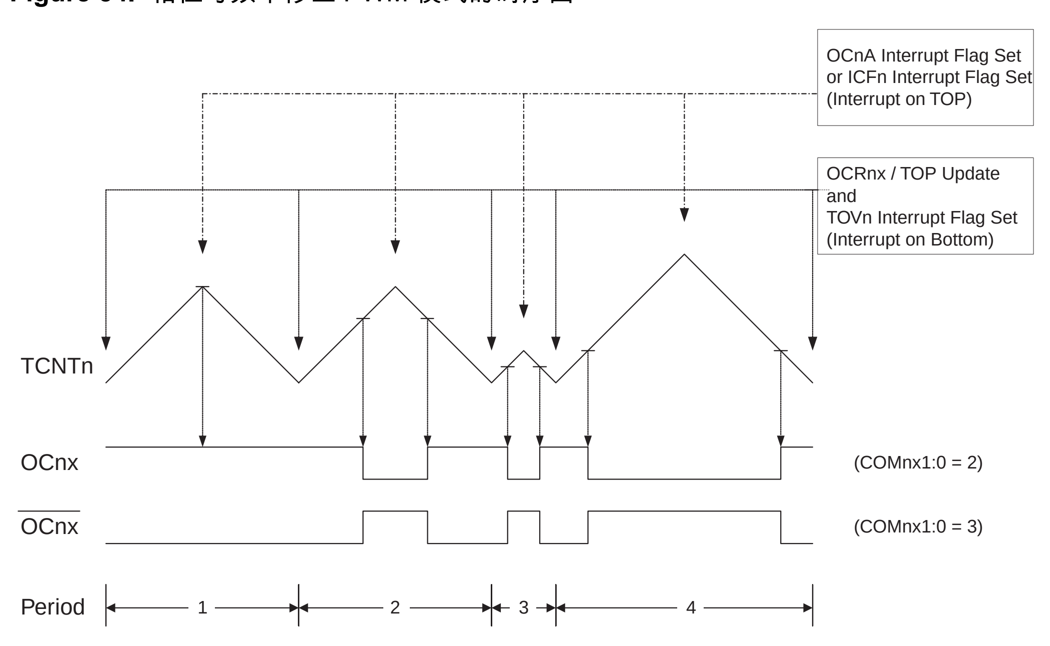
相频修正修正 PWM 模式与相位修正 PWM 模式的主要区别在于 OCRnx 寄存器的更新时间，详见 Figure 53 与 Figure 54。

相频修正修正 PWM 模式的 PWM 分辨率可由 ICRn 或 OCRnA 定义。最小分辨率为 2 比特 (ICRn 或 OCRnA 设为 0x0003)，最大分辨率为 16 位 (ICRn 或 OCRnA 设为 MAX)。PWM 分辨率位数可用下式计算：

$$R_{PFCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

工作于相频修正PWM模式时，计数器的数值一直累加到ICRn (WGMn3:0 = 8) 或OCRnA (WGMn3:0 = 9)，然后改变计数方向。在一个定时器时钟里TCNTn值等于TOP值。具体的时序图为 Figure 54。图中给出了当使用 OCRnA 或 ICRn 来定义 TOP 值时的相频修正PWM模式。图中柱状的TCNTn表示这是双边斜坡操作。方框图同时包含了普通的PWM输出以及反向 PWM 输出。TCNTn 斜坡上的短水平线表示 OCRnx 和 TCNTn 的匹配比较。比较匹配发生时， OCnx 中断标志将被置位。

Figure 54. 相位与频率修正 PWM 模式的时序图



在 OCRnx 寄存器通过双缓冲方式得到更新的同一个时钟周期里 T/C 溢出标志 TOVn 置位。若 TOP 由 OCRnA 或 ICRn 定义，则当 TCNTn 达到 TOP 值时 OCnA 或 ICFn 置位。这些中断标志位可用来在每次计数器达到 TOP 或 BOTTOM 时产生中断。

改变TOP值时必须保证新的TOP值不小于所有比较寄存器的数值。否则TCNTn与OCRnx不会产生比较匹配。

如 Figure 54 所示，与相位修正模式形成对照的是，相频修正 PWM 模式生成的输出在所有的周期中均为对称信号。这是由于 OCRnx 在 BOTTOM 得到更新，上升与下降斜坡长度始终相等。因此输出脉冲为对称的，确保了频率是正确的。

使用固定 TOP 值时最好使用 ICRn 寄存器定义 TOP。这样 OCRnA 就可以用于在 OCnA 输出 PWM 波。但是，如果 PWM 基频不断变化 (通过改变 TOP 值)， OCRnA 的双缓冲特性使其更适合于这个应用。

工作于相频修正 PWM 模式时，比较单元可以在 OCnx 引脚上输出 PWM 波形。设置 COMnx1:0 为 2 可以产生普通的 PWM 信号；为 3 则可以产生反向 PWM 波形。(参见 P 121Table 60)。要想真正输出信号还必须将OCnx的数据方向设置为输出。产生PWM波形的机理是 OCnx 寄存器在 OCRnx 与升序记数的 TCNTn 匹配时置位 (或清零) ,与降序记数的 TCNTn 匹配时清零 (或置位)。输出的 PWM 频率可以通过如下公式计算得到：

$$f_{OCnxPFCPWM} = \frac{f_{clk_I/O}}{2 \cdot N \cdot TOP}$$

变量 N 代表分频因子 (1、8、64、 256 或 1024)。

OCRnx 寄存器处于极值时说明了相频修正 PWM 模式的一些特殊情况。在普通 PWM 模式下，若 OCRnx 等于 BOTTOM，输出一直保持为低电平；若 OCRnx 等于 TOP，则输出保持为高电平。反向 PWM 模式则正好相反

若 OCnA 作为 TOP 值 (WGMn3:0 = 9) 且 COMnA1:0 = 1， OCnA 输出占空比为 50%。

定时器 / 计数器时序图

定时器 / 计数器为同步电路，因而时钟 clk_{Tn} 表示为时钟使能信号。图中说明了何时设置中断标志及何时使用 OCRnx 缓冲器中的数据更新 OCRnx 寄存器 (工作于双缓冲器模式时)。 Figure 55 给出了置位 OCFnx 的时序图。

Figure 55. T/C 时序图 ， OCFnx 置位，无预分频器

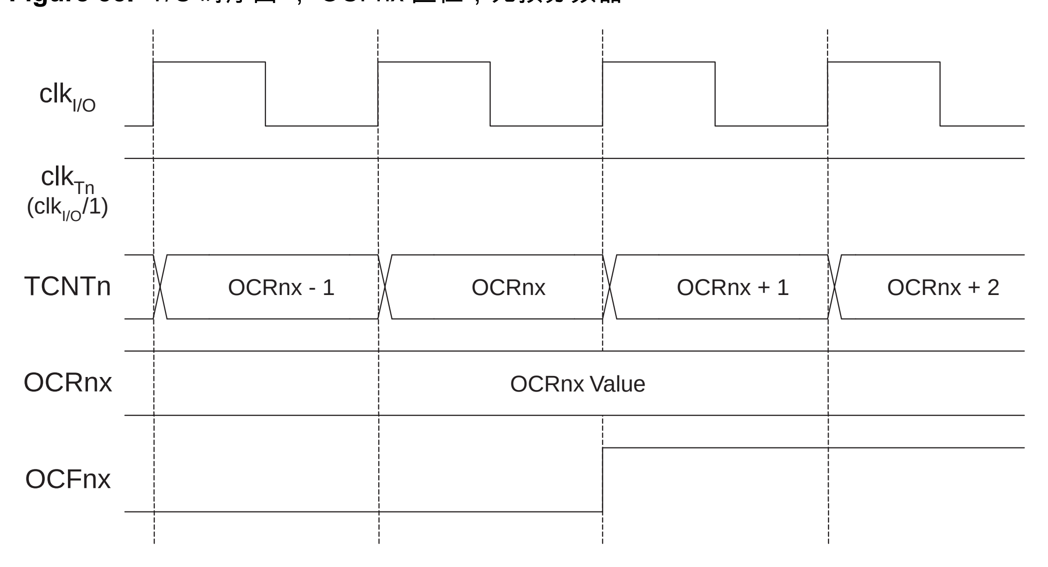


Figure 56 给出相同的时钟数据，但预分频使能。

Figure 56. T/C 时序图，置位 OCFnx，预分频器为 $f_{clk_I/O}/8$

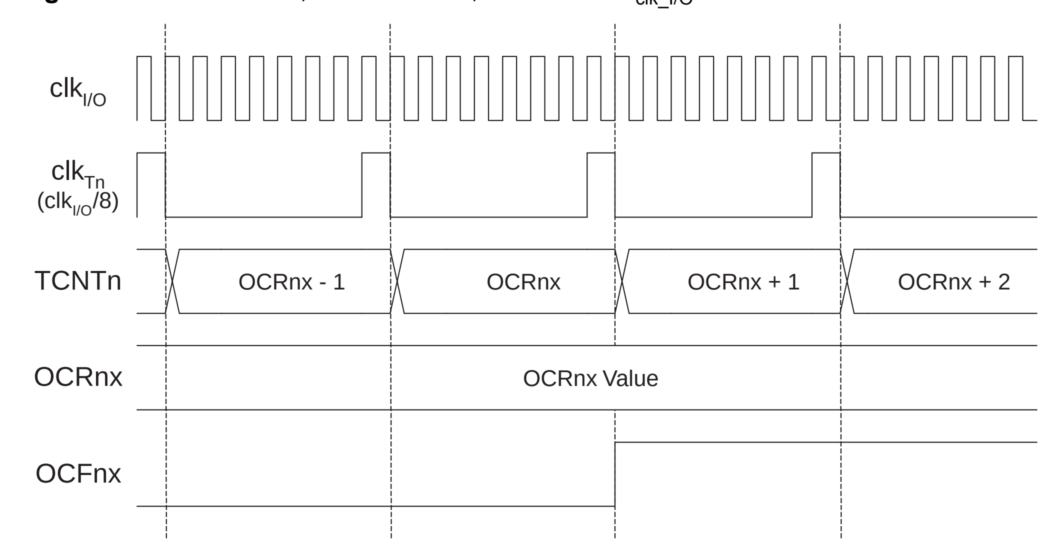


Figure 57 给出工作在不同模式下接近 TOP 值时的计数序列。工作于相频修正 PWM 模式时， OCRnx 寄存器在 BOTTOM 被更新。时序图相同，但 TOP 需要用 BOTTOM 代替，BOTTOM+1 代替 TOP-1，等等。同样的命名规则也适用于那些在 BOTTOM 置位 TOVn 标志的工作模式。

Figure 57. T/C 时序图，无预分频器

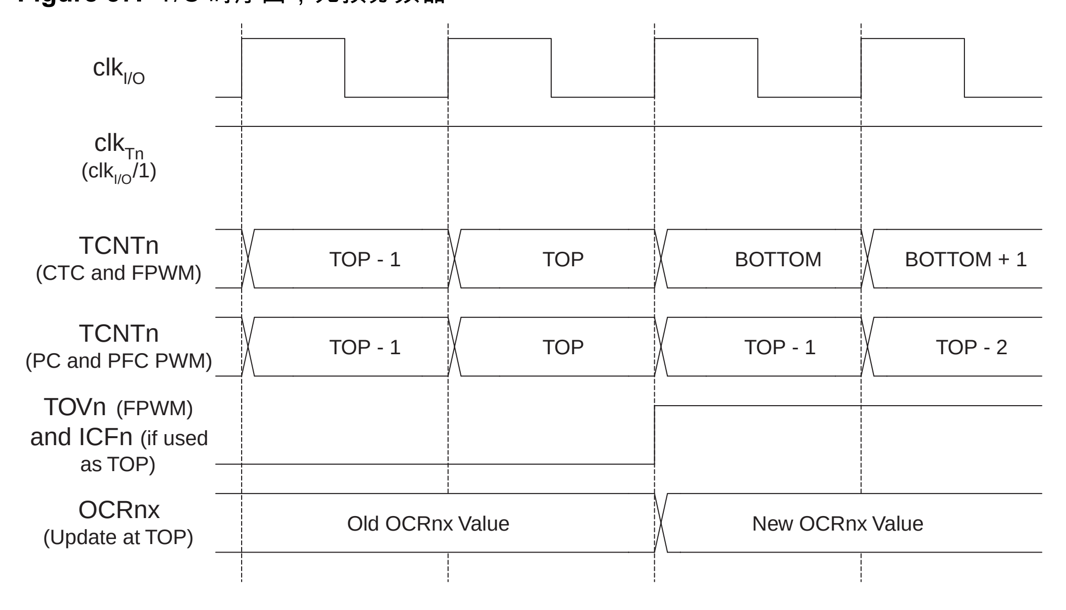
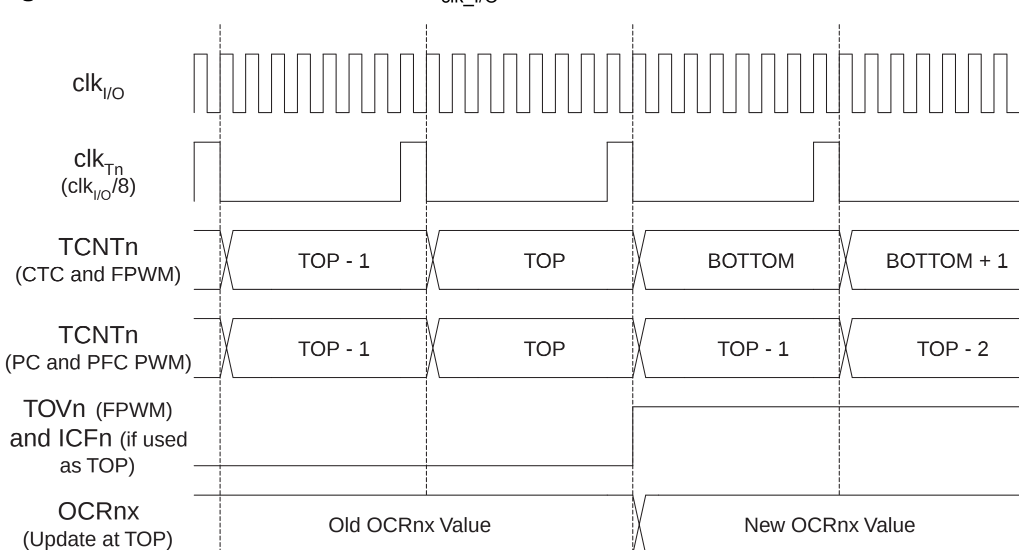


Figure 58 给出相同的时钟数据，但预分频使能。

Figure 58. T/C 时序图，预分频器为 $f_{clk_I/O}/8$



16 位定时器 / 计数器寄存器说明

定时器 / 计数器 1 控制寄存器 A - TCCR1A

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

定时器 / 计数器 3 控制寄存器 A - TCCR3A

Bit	7	6	5	4	3	2	1	0	
	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	TCCR3A
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7:6 – COMnA1:0: 通道 A 的比较输出模式**
- **Bit 5:4 – COMnB1:0: 通道 B 的比较输出模式**
- **Bit 3:2 – COMnC1:0: 通道 C 的比较输出模式**

COMnA1:0、COMnB1:0 与 COMnC1:0 分别控制 OCnA 、OCnB 与 OCnC 的状态。如果 COMnA1:0(COMnB1:0或COMnC1:0)的一位或两位被写入"1” ,OCnA(OCnB或OCnC) 输出功能将取代 I/O 端口功能。此时 OCnA(OCnB 或 OCnC) 相应的输出引脚数据方向控制必须置位以使能输出驱动器。

OCnA(OCnB 或 OCnC) 与物理引脚相连时 ,COMnx1:0 的功能由 WGMn3:0 的设置决定。Table 58 给出当 WGMn3:0 设置为普通模式与 CTC 模式 (非 PWM) 时 COMnx1:0 的功能定义。

Table 58. 比较输出模式，非 PWM

COMnA1/COMnB1/ COMnC1	COMnA0/COMnB0/ COMnC0	说明
0	0	普通端口操作， OCnA/OCnB/OCnC 未连接
0	1	比较匹配时 OCnA/OCnB/OCnC 电平取反
1	0	比较匹配时清零 OCnA/OCnB/OCnC(输出低电平)
1	1	比较匹配时置位 OCnA/OCnB/OCnC(输出高电平)

Table 59 给出 WGMn3:0 设置为快速 PWM 模式时 COMnx1:0 的功能定义。

Table 59. 比较输出模式，快速 PWM

COMnA1/COMnB1/ COMnC0	COMnA0/COMnB0/ COMnC0	说明
0	0	普通端口操作， OCnA/OCnB/OCnC 未连接
0	1	WGMn3=0: 普通端口操作， OCnA/OCnB/OCnC 未连接 WGMn3=1: 比较匹配时 OCnA 电平取反， OCnB/OCnC 保留
1	0	比较匹配时清零 OCnA/OCnB/OCnC，在 TOP 时置位 OCnA/OCnB/OCnC
1	1	比较匹配时置位 OCnA/OCnB/OCnC，在 TOP 时清零 OCnA/OCnB/OCnC

Note: 当OCRnA/OCRnB/OCRnC等于TOP且COMnA1/COMnB1/COMnC1 置位时，比较匹配被忽略，但 OCnA/OCnB/OCnC 的置位 / 清零操作有效。详见 P 113 " 快速 PWM 模式 " 。

Table 59给出当WGMn3:0设置为相位修正PWM模式或相频修正PWM模式时COMnx1:0的功能定义。

Table 60. 比较输出模式，相位修正及相频修正 PWM 模式

COMnA1/COMnB/ COMnC1	COMnA0/COMnB0/ COMnC0	说明
0	0	普通端口操作， OCnA/OCnB/OCnC 未连接
0	1	WGMn3=0: 普通端口操作， OCnA/OCnB/OCnC 未连接 WGMn3=1: 比较匹配时 OCnA 电平取反， OCnB/OCnC 保留
1	0	升序记数时比较匹配将清零 OCnA/OCnB/OCnC ，降序记数时比较匹配将置位 OCnA/OCnB/OCnC
1	1	升序记数时比较匹配将置位 OCnA/OCnB/OCnC，降序记数时比较匹配将清零 OCnA/OCnB/OCnC

Note: OCRnA/OCRnB/OCRnC等于TOP且COMnA1/COMnB1/COMnC1置位是一个特殊情况。详细信息请参见 P 114 " 相位修正 PWM 模式 " 。

- **Bit 1:0 – WGMn1:0: 波形发生模式**

这两位与位于 TCCRnB 寄存器的 WGMn3:2 相结合，用于控制计数器的计数序列——计数器计数的上限值和确定波形发生器的工作模式 (见 Table 61)。T/C 支持的工作模式有：普通模式 (计数器)，比较匹配时清零定时器 (CTC) 模式，及三种脉宽调制 (PWM) 模式 (P 111 " 工作模式 ")。

Table 61. 波形产生模式的位描述

模式	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	定时器 / 计数器工作模式 (1)	TOP	OCRnx 更新时刻	TOVn 置位时刻
0	0	0	0	0	普通模式	0xFFFF	立即更新	MAX
1	0	0	0	1	8 位相位修正 PWM	0x00FF	TOP	BOTTOM
2	0	0	1	0	9 位相位修正 PWM	0x01FF	TOP	BOTTOM
3	0	0	1	1	10 位相位修正 PWM	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	立即更新	MAX
5	0	1	0	1	8 位快速 PWM	0x00FF	TOP	TOP
6	0	1	1	0	9 位快速 PWM	0x01FF	TOP	TOP
7	0	1	1	1	10 位快速 PWM	0x03FF	TOP	TOP
8	1	0	0	0	相位与频率修正 PWM	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	相位与频率修正 PWM	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	相位修正 PWM	ICRn	TOP	BOTTOM
11	1	0	1	1	相位修正 PWM	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	立即更新	MAX
13	1	1	0	1	保留	–	–	–
14	1	1	1	0	快速 PWM	ICRn	TOP	TOP
15	1	1	1	1	快速 PWM	OCRnA	TOP	TOP

Note: 1. CTCn 和 PWMn1:0 的定义已经不再使用了，要使用 WGMn2:0。但是两个版本的功能和位置是兼容的。

定时器 / 计数器 1 控制寄存器 B - TCCR1B

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
读 / 写	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

定时器 / 计数器 3 控制寄存器 B - TCCR3B

Bit	7	6	5	4	3	2	1	0	
	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
读 / 写	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – ICNCn: 输入捕捉噪声抑制器**

置位 ICNC1 将使能输入捕捉噪声抑制功能。此时外部引脚 ICPn 的输入被滤波。其作用是从 ICPn 引脚连续进行 4 次采样。如果 4 个采样值都相等，那么信号送入边沿检测器。因此使能该功能使得输入捕捉被延迟了 4 个时钟周期。

- **Bit 6 – ICESn: 输入捕捉触发沿选择**

该位选择使用 ICPn 上的哪个边沿触发捕获事件。ICESn 为 "0" 选择的是下降沿触发输入捕捉； ICESn 为 "1" 选择的是逻辑电平的上升沿触发输入捕捉。

按照 ICESn 的设置捕获到一个事件后，计数器的数值被复制到 ICRn 寄存器。捕获事件还会置为 ICFn。如果此时中断使能，输入捕捉事件即被触发。

当 ICRn 用作 TOP 值 (见 TCCRnA 与 TCCRnB 寄存器中 WGMn3:0 位的描述) 时，ICPn 与输入捕捉功能脱开，从而输入捕捉功能被禁用。

- **Bit 5 – 保留位**

该位保留。为保证与将来器件的兼容性，写 TCCRnB 时，该位必须写入 "0"。

- **Bit 4:3 – WGMn3:2: 波形发生模式**

见 TCCRnA 寄存器中的描述。

- **Bit 2:0 – CSn2:0: 时钟选择**

这 3 位用于选择 T/C 的时钟源，见 Figure 55 与 Figure 56。



Table 62. 时钟选择位描述

CSn2	CSn1	CSn0	说明
0	0	0	无时钟源 (T/C 停止)
0	0	1	$clk_{I/O}$/1 (无预分频)
0	1	0	$clk_{I/O}$/8 (来自预分频器)
0	1	1	$clk_{I/O}$/64 (来自预分频器)
1	0	0	$clk_{I/O}$/256 (来自预分频器)
1	0	1	$clk_{I/O}$/1024 (来自预分频器)
1	1	0	外部 Tn 引脚，下降沿驱动
1	1	1	外部 Tn 引脚，上升沿驱动

选择使用外部时钟源后，即使 Tn 引脚被定义为输出，其 n 引脚上的逻辑信号电平变化仍然会驱动 T/Cn 计数，这个特性允许用户通过软件来控制计数。

定时器 / 计数器 1 控制寄存器 C - TCCR1C

Bit	7	6	5	4	3	2	1	0	
	FOC1A	FOC1B	FOC1C	–	–	–	–	–	TCCR1C
读 / 写	W	W	W	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	

定时器 / 计数器 3 控制寄存器 C - TCCR3C

Bit	7	6	5	4	3	2	1	0	
	FOC3A	FOC3B	FOC3C	–	–	–	–	–	TCCR3C
读 / 写	W	W	W	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – FOCnA: 强制输出比较通道 A**
- **Bit 6 – FOCnB: 强制输出比较通道 B**
- **Bit 5 – FOCnC: 强制输出比较通道 C**

FOCnA/FOCnB/FOCnC 位只在 WGMn3:0 位被设置为非 PWM 模式时才有效。对 FOCnA/FOCnB/FOCnC 写 "1" 将强制波形发生器产生一次成功的比较匹配，并使波形发生器依据 COMnx1:0 的设置而改变 OCnA/OCnB/OCnC 的输出状态。FOCnA/FOCnB/FOCnC 的作用如同一个选通信号，COMnx1:0 的设置才是最终确定比较匹配结果的因素。

FOCnA/FOCnB/FOCnC 选通信号不会产生任何中断请求，也不会对计数器清零，象使用 OCRnA 为 TOP 值的 CTC 工作模式那样。

FOCnA/FOCnB/FOCnC 的读返回值总为零。

- **Bit 4:0 – 保留位**

这几位保留。为保证与将来器件的兼容性，写 TCCRnC 时，这几位必须写入 "0"。

定时器 / 计数器 1 － TCNT1H 和 TCNT1L

Bit	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1[7:0]								TCNT1L
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

定时器 / 计数器 3 － TCNT3H 和 TCNT3L

Bit	7	6	5	4	3	2	1	0	
	TCNT3[15:8]								TCNT3H
	TCNT3[7:0]								TCNT3L
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

TCNTnH与TCNTnL组成了T/Cn的数据寄存器TCNTn。通过它们可以直接对定时器/计数器单元的 16 位计数器进行读写访问。为保证 CPU 对高字节与低字节的同时读写，必须使用一个 8 位临时高字节寄存器 TEMP。 TEMP 是所有的 16 位寄存器共用的，详见 P 102 " 访问 16 位寄存器 " 。

在计数器运行期间修改TCNTn的内容有可能丢失一次TCNTn与OCRnx的比较匹配操作。

写 TCNTn 寄存器将在下一个定时器周期阻塞比较匹配。

输出比较寄存器 1 A － OCR1AH 和 OCR1AL

Bit	7	6	5	4	3	2	1	0	
	OCR1A[15:8]								OCR1AH
	OCR1A[7:0]								OCR1AL
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器 1 B － OCR1BH 和 OCR1BL

Bit	7	6	5	4	3	2	1	0	
	OCR1B[15:8]								OCR1BH
	OCR1B[7:0]								OCR1BL
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

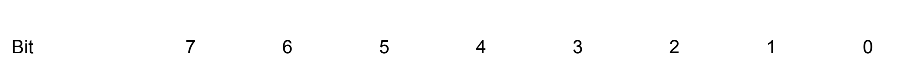
输出比较寄存器 1 C － OCR1CH 和 OCR1CL

Bit	7	6	5	4	3	2	1	0	
	OCR1C[15:8]								OCR1CH
	OCR1C[7:0]								OCR1CL
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器 3 A － OCR3AH 和 OCR3AL

Bit	7	6	5	4	3	2	1	0	
	OCR3A[15:8]								OCR3AH
	OCR3A[7:0]								OCR3AL
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器 3 B － OCR3BH 和 OCR3BL



	OCR3B[15:8]								OCR3BH
	OCR3B[7:0]								OCR3BL
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

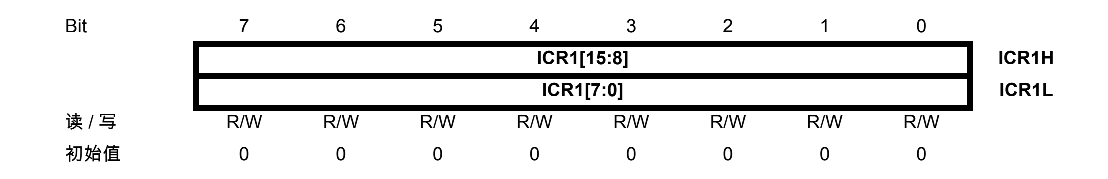
输出比较寄存器 3 C - OCR3CH 和 OCR3CL

Bit	7	6	5	4	3	2	1	0	
	OCR3C[15:8]								OCR3CH
	OCR3C[7:0]								OCR3CL
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

该寄存器中的 16 位数据与 TCNTn 寄存器中的计数值进行连续的比较，一旦数据匹配，将产生一个输出比较中断，或改变 OCnx 的输出逻辑电平。

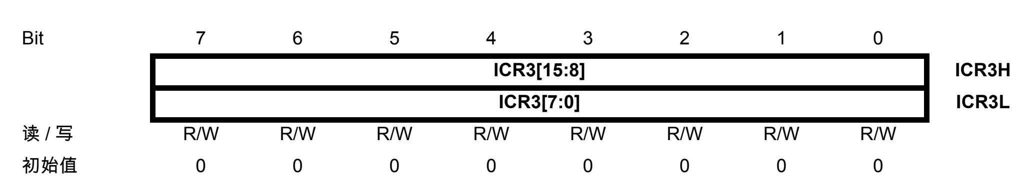
输出比较寄存器长度为 16 位。为保证 CPU 对高字节与低字节的同时读写，必须使用一个 8 位临时高字节寄存器 TEMP。 TEMP 是所有的 16 位寄存器共用的，详见 P 102 “ 访问 16 位寄存器 ” 。

输入捕捉寄存器 1 - ICR1H 和 ICR1L



Bit	7	6	5	4	3	2	1	0	
	ICR1[15:8]								ICR1H
	ICR1[7:0]								ICR1L
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输入捕捉寄存器 3 - ICR3H 和 ICR3L



Bit	7	6	5	4	3	2	1	0	
	ICR3[15:8]								ICR3H
	ICR3[7:0]								ICR3L
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

当外部引脚 ICPn(或 T/C1 的模拟比较器) 有输入捕捉触发信号产生时 , 计数器 TCNTn 中的值写入 ICR1 中。 ICR1 的设定值可作为计数器的 TOP 值。

输入捕捉寄存器长度为 16 位。为保证 CPU 对高字节与低字节的同时读写，必须使用一个 8 位临时高字节寄存器 TEMP。 TEMP 是所有的 16 位寄存器共用的，详见 P 102 “ 访问 16 位寄存器 ” 。

定时器 / 计数器中断屏蔽寄存器 - TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

Note: 该寄存器包含几个 T/C 的中断控制位 , 但本节中只对 T1 位进行说明 , 其余位将在各自的小节中加以说明。

- **Bit 5 – TICIE1: T/C1 输入捕捉中断使能**

当该位被设为 "1”，且状态寄存器中的 I 位被设为 "1” 时， T/C1 的输入捕捉中断使能。一旦 TIFR1 的 ICF1 置位 , CPU 即开始执行 T/C1 输入捕捉中断服务程序 (见 P 55 “ 中断 ”)。

- **Bit 4 – OCIE1A:T/C1 输出比较 A 匹配中断使能**

当该位被设为 "1"，且状态寄存器中的 I 位被设为 "1" 时，使能 T/C1 的输出比较 A 匹配中断使能。一旦 TIFR1 上的 OCF1A 置位， CPU 即开始执行 T/C1 输出比较 A 匹配中断服务程序 (见 P 55 " 中断 ")。

• **Bit 3 – OCIE1B:T/C1 输出比较 B 匹配中断使能**

当该位被设为 "1"，且状态寄存器中的 I 位被设为 "1" 时，使能 T/C1 的输出比较 B 匹配中断使能。一旦 TIFR1 上的 OCF1B 置位， CPU 即开始执行 T/C1 输出比较 B 匹配中断服务程序 (见 P 55 " 中断 ")。

• **Bit 2 – TOIE1:T/C1 溢出中断使能**

当该位被设为 "1"，且状态寄存器中的 I 位被设为 "1" 时， T/C1 的溢出中断使能。一旦 TIFR 上的 TOV1 置位， CPU 即开始执行 T/C1 溢出中断服务程序 (见 P 55 " 中断 ")。

扩展的定时器 / 计数器中断屏蔽寄存器 - ETIMSK

Bit	7	6	5	4	3	2	1	0	
	–	–	TICIE3	OCIE3A	OCIE3B	TOIE3	OCIE3C	OCIE1C	ETIMSK
读 / 写	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

Note: 该寄存器在 ATmega103 兼容模式下无效。

• **Bit 7:6 – 保留位**

这两位保留。为保证与将来器件的兼容性，写 ETIMSK 时，这两位必须写入 "0"。

• **Bit 5 – TICIE3:T/C3, 输入捕捉中断使能**

当该位被设为 "1"，且状态寄存器中的 I 位被设为 "1" 时， T/C3 的输入捕捉中断使能。一旦 ETIFR 的 ICF3 置位 ,CPU 即开始执行 T/C3 输入捕捉中断服务程序 (见 P 55 " 中断 ")。

• **Bit 4 – OCIE3A:T/C3 输出比较 A 匹配中断使能**

当该位被设为 "1"，且状态寄存器中的 I 位被设为 "1" 时，T/C3 的输出比较 A 匹配中断使能。一旦 ETIFR 上的 OCF3A 置位， CPU 即开始执行 T/C3 输出比较 A 匹配中断服务程序 (见 P 55 " 中断 ")。

• **Bit 3 – OCIE3B:T/C3 输出比较 B 匹配中断使能**

当该位被设为 "1"，且状态寄存器中的 I 位被设为 "1" 时，T/C3 的输出比较 B 匹配中断使能。一旦 ETIFR 上的 OCF3B 置位， CPU 即开始执行 T/C3 输出比较 B 匹配中断服务程序 (见 P 55 " 中断 ")。

• **Bit 2 – TOIE3:T/C3 溢出中断使能**

当该位被设为 "1"，且状态寄存器中的 I 位被设为 "1" 时， T/C3 的溢出中断使能。一旦 ETIFR 上的 TOV3 置位， CPU 即开始执行 T/C3 溢出中断服务程序 (见 P 55 " 中断 ")。

• **Bit 1 – OCIE3C:T/C3 输出比较 C 匹配中断使能**

当该位被设为 "1"，且状态寄存器中的 I 位被设为 "1" 时，T/C3 的输出比较 C 匹配中断使能。一旦 ETIFR 上的 OCF3C 置位， CPU 即开始执行 T/C3 输出比较 C 匹配中断服务程序 (见 P 55 " 中断 ")。

• **Bit 0 – OCIE1C:T/C1 输出比较 C 匹配中断使能**

当该位被设为 "1"，且状态寄存器中的 I 位被设为 "1" 时，T/C1 的输出比较 C 匹配中断使能。一旦 ETIFR 上的 OCF1C 置位， CPU 即开始执行 T/C1 输出比较 C 匹配中断服务程序 (见 P 55 " 中断 ")。



定时器 / 计数器中断标志寄存器 - TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

Note: 该寄存器包含几个 T/C 的标志位，但本节中只对 T1 位进行说明，其余位将在各自的小节中加以说明。

- **Bit 5 – ICF1:T/C1 输入捕捉标志位**

外部引脚 ICP1 出现捕捉事件时 ICF1 置位。此外，当 ICR1 作为计数器的 TOP 值时，一旦计数器值达到 TOP， ICF1 也置位。

执行输入捕捉中断服务程序时 ICF1 自动清零。也可以对其写入逻辑 "1" 来清除该标志位。

- **Bit 4 – OCF1A:T/C1 输出比较 A 匹配标志位**

当 TCNT1 与 OCR1A 匹配成功时，该位被设为 "1"。

强制输出比较 (FOC1A) 不会置位 OCF1A。

执行强制输出比较匹配 A 中断服务程序时 OCF1A 自动清零。也可以对其写入逻辑 "1" 来清除该标志位。

- **Bit 3 – OCF1B:T/C1 输出比较 B 匹配标志位**

当 TCNT1 与 OCR1B 匹配成功时，该位被设为 "1"。

强制输出比较 (FOC1B) 不会置位 OCF1B。

执行强制输出比较匹配 B 中断服务程序时 OCF1B 自动清零。也可以对其写入逻辑 "1" 来清除该标志位。

- **Bit 2 – TOV1:T/C1 溢出标志**

该位的设置与 T/C1 的工作方式有关。工作于普通模式和 CTC 模式时，T/C1 溢出时 TOV1 置位。对工作在其它模式下的 TOV1 标志位置位，见 P 122Table 61 。

执行溢出中断服务程序时 OCF1A 自动清零。也可以对其写入逻辑 "1" 来清除该标志位。

扩展的定时器 / 计数器中断标志寄存器 - ETIFR

Bit	7	6	5	4	3	2	1	0	
	–	–	ICF3	OCF3A	OCF3B	TOV3	OCF3C	OCF1C	ETIFR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7:6 – 保留位**

这两位保留。为保证与将来器件的兼容性，写 ETIFR 时，这两位必须写入 "0"。

- **Bit 5 – ICF3:T/C3 输入捕捉标志位**

外部引脚 ICP3 出现捕捉事件时 ICF3 置位。此外，当 ICR3 作为计数器的 TOP 值时，一旦计数器值达到 TOP， ICF3 也置位。

执行输入捕捉中断服务程序时 ICF3 自动清零。也可以对其写入逻辑 "1" 来清除该标志位。

- **Bit 4 – OCF3A:T/C3 输出比较 A 匹配标志位**

当 TCNT3 与 OCR3A 匹配成功时，该位被设为 "1"。

强制输出比较 (FOC3A) 不会置位 OCF3A。

执行强制输出比较匹配 3A 中断服务程序时 OCF3A 自动清零。也可以对其写入逻辑 "1" 来清除该标志位。

- **Bit 3 – OCF3B:T/C3 输出比较 B 匹配标志位**

当 TCNT3 与 OCR3B 匹配成功时，该位被设为 "1"。

强制输出比较 (FOC3B) 不会置位 OCF3B。

执行强制输出比较匹配 3B 中断服务程序时 OCF3B 自动清零。也可以对其写入逻辑 "1" 来清除该标志位。

- **Bit 2 – TOV3:T/C3 溢出标志**

该位的设置与 T/C3 的工作方式有关。工作于普通模式和 CTC 模式时 ,T/C3 溢出时 TOV3 置位。对工作在其它模式下的 TOV3 标志位置位，见 P 95Table 52 。

执行溢出中断服务程序时 OCF3B 自动清零。也可以对其写入逻辑 "1" 来清除该标志位。

- **Bit 1 – OCF3C:T/C3 输出比较 C 匹配标志位**

当 TCNT3 与 OCR3C 匹配成功时，该位被设为 "1"。

强制输出比较 (FOC3C) 不会置位 OCF3C。

执行强制输出比较匹配 3C 中断服务程序时 OCF3C 自动清零。也可以对其写入逻辑 "1" 来清除该标志位。

- **Bit 0 – OCF1C:T/C1 输出比较 C 匹配标志位**

当 TCNT1 与 OCR1C 匹配成功时，该位被设为 "1"。

强制输出比较 (FOC1C) 不会置位 OCF1C。

执行强制输出比较匹配 1 C 中断服务程序时 OCF1C 自动清零。也可以对其写入逻辑 "1" 来清除该标志位。





定时器 / 计数器 3、定时器 / 计数器 2 和定时器 / 计数器 1 的预分频器

T/C3、T/C1 与 T/C2 共用一个预分频模块 , 但它们可以有不同的分频设置。下述内容适用于 T/C3、 T/C1 与 T/C2。

内部时钟源

当 CSn2:0 = 1 时，系统内部时钟直接作为 T/C 的时钟源，这也是 T/C 最高频率的时钟源 $f_{CLK_I/O}$，与系统时钟频率相同。预分频器可以输出 4 个不同的时钟信号 $f_{CLK_I/O}/8$、$f_{CLK_I/O}/64$、$f_{CLK_I/O}/256$ 或 $f_{CLK_I/O}/1024$。

预分频器复位

预分频器是独立运行的。也就是说，其操作独立于 T/C 的时钟选择逻辑，且它由 T/C1、T/C2与T/C3共享。由于预分频器不受T/C时钟选择的影响 , 预分频器的状态需要包含预分频时钟被用到何处这样的信息。一个典型的例子发生在定时器使能并由预分频器驱动 (6 > CSn2:0 > 1) 的时候 : 从计时器使能到第一次开始计数可能花费 1 到N+1个系统时钟周期，其中 N 等于预分频因子 (8、 64、 256 或 1024)。

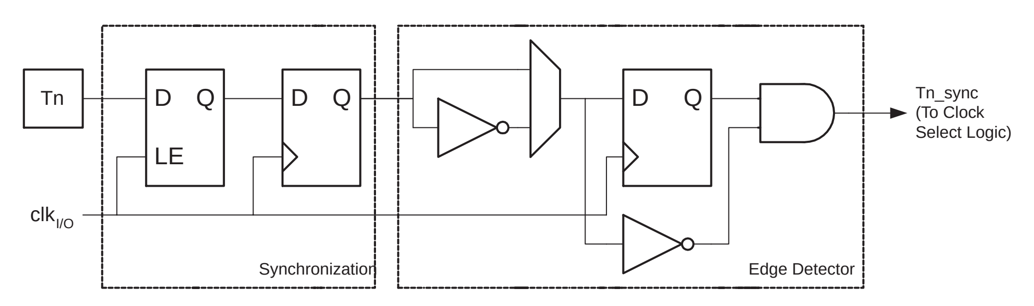
通过复位预分频器来同步 T/C 与程序运行是可能的。但是必须注意另一个 T/C 是否也在使用这一预分频器，因为预分频器复位将会影响所有与其连接的 T/C。

外部时钟源

由 Tn 引脚提供的外部时钟源可以用作 T/C 时钟 $clk_{T1}/clk_{T2}/clk_{T3}$。引脚同步逻辑在每个系统时钟周期对引脚 Tn 进行采样。然后将同步 (采样) 信号送到边沿检测器。Figure 59 给出了 Tn 同步采样与边沿检测逻辑的功能等效方框图。寄存器由内部系统时钟 $clk_{I/O}$ 的上跳沿驱动。当内部时钟为高时，锁存器可以看作时透明的。

CSn2:0 = 7 时边沿检测器检测到一个正跳变产生一个 $clk_{T1}/clk_{T2}/clk_{T3}$ 脉冲； CSn2:0 = 6 时一个负跳变就产生一个 clk_{T0} 脉冲。

Figure 59. Tn 引脚采样



由于引脚上同步与边沿监测电路的存在，引脚 Tn 上的电平变化需要延时 2.5 到 3.5 个系统时钟周期才能使计数器进行更新。

禁止或使能时钟输入必须在 Tn 保持稳定至少一个系统时钟周期后才能进行，否则有产生错误 T/C 时钟脉冲的危险。

为保证正确的采样，外部时钟脉冲宽度必须大于一个系统时钟周期。在占空比为 50% 时外部时钟频率必须小于系统时钟频率的一半 ($f_{ExtClk} < f_{clk_I/O}/2$)。由于边沿检测器使用的是采样这一方法，它能检测到的外部时钟最多是其采样频率的一半 (Nyquist 采样定理)。然而，由于振荡器 (晶体、谐振器与电容) 本身误差带来的系统时钟频率及占空比的差异，建议外部时钟的最高频率不要大于 $f_{clk_I/O}/2.5$。

外部时钟源不送入预分频器。

Figure 60. T/C1 、 T/C2 与 T/C3 的预分频器

Note: 输入引脚 (T3/T2/T1) 的同步逻辑见 Figure 59。

特殊功能 IO 寄存器 - SFIOR

Bit	7	6	5	4	3	2	1	0	
	TSM	**–**	**–**	**–**	**ACME**	**PUD**	**PSR0**	**PSR321**	**SFIOR**
读 / 写	R/W	R	R	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – TSM:T/C 同步模式**

只要 TSM 置位，PSR0 与 PSR321 的数值将保持不变，使得相关的定时器 / 计数器预分频器处于持续复位状态。这样相关的 T/C 将停止工作。用户可以为它们赋予相同的数值而不会出现在配置一个定时器 / 计数器时另一个 T/C 在运行的现象。一旦 TSM 清零，相关的定时器 / 计数器同时开始计数。

- **Bit 0 – PSR321: T/C3、 T/C2 与 T/C1 预分频器复位**

置位时 T/C3、 T/C2 与 T/C1 的预分频器复位。操作完成后这一位通常由硬件立即清零，除非 TSM 置位。要注意的是 T/C3、 T/C2 与 T/C1 共用一个预分频器，复位对两个计时器都有影响。该位总是读为 "0”。

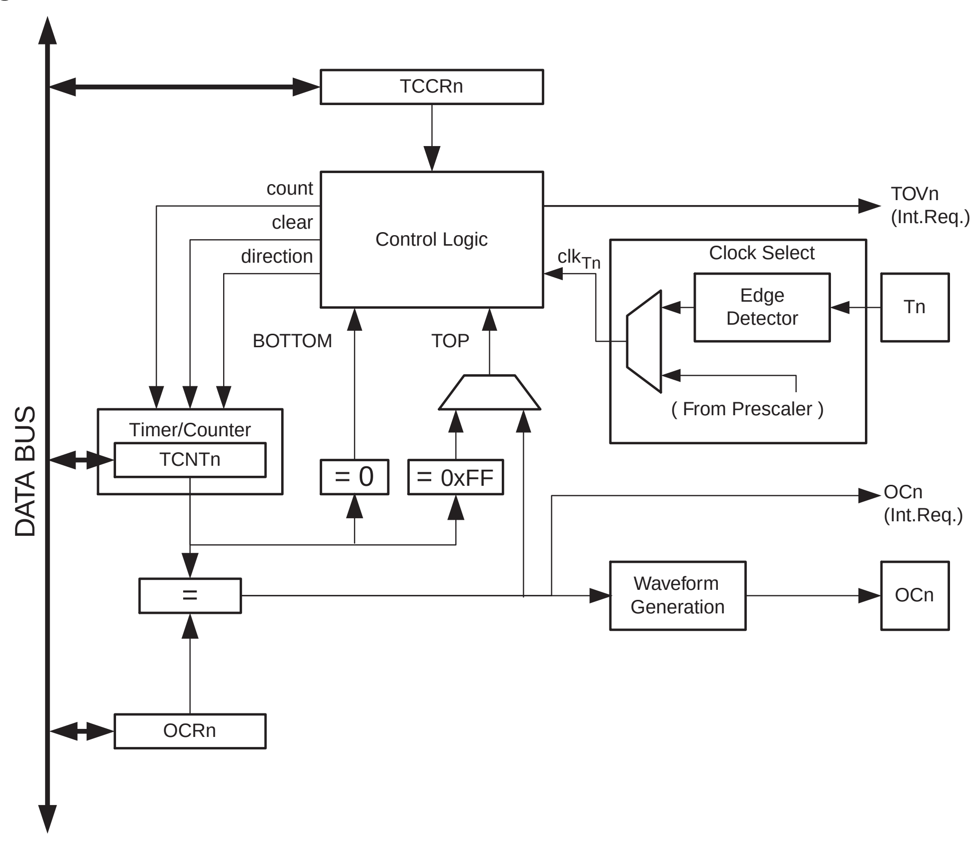
具有 PWM 功能的 8 位定时器 / 计数器 2

T/C2 是一个通用单通道 8 位定时 / 计数器，其主要特点如下：
- **单通道计数器**
- **比较匹配时清零定时器 (自动重载)**
- **无干扰脉冲，相位正确的脉宽调制器 (PWM)**
- **频率发生器**
- **外部事件计数器**
- **10 位时钟预分频器**
- **溢出与比较匹配中断源 (TOV2 与 OCF2)**

综述

Figure 61为8位T/C的方框图。实际的引脚图请参见 P 2"引脚配置" 。CPU可访问的I/O寄存器，包括 I/O 位和 I/O 引脚以粗体表示。器件具体 I/O 寄存器与位定义见 P 142"8 位定时器 / 计数器寄存器说明 " 。

Figure 61. 8 位 T/C 方框图



寄存器

定时器 / 计数器 TCNT2、输出比较寄存器 OCR2 为 8 位寄存器。中断请求 (图中简写为 Int.Req.)。信号在定时器中断标志寄存器 TIFR 都有反映。所有中断都可以通过定时器中断屏蔽寄存器 TIMSK 单独进行屏蔽。图中未给出 TIFR 与 TIMSK。

T/C的时钟可以为通过预分频器的内部时钟或通过外部时钟源T2引脚接入。时钟选择逻辑模块控制引起 T/C 计数值增加 (或减少) 的时钟源。没有选择时钟源时 T/C 处于停止状态。时钟选择逻辑模块的输出称为 clk_{T0}。

双缓冲的输出比较寄存器 OCR2 一直与 T/C 的数值进行比较。波形发生器利用比较结果产生 PWM 波形或在比较输出引脚 OC2 输出可变频率的信号。参见 P 134 " 输出比较单元 " 。比较匹配结果还会置位比较匹配标志 OCF2，用来产生输出比较中断请求。

定义

本文的许多寄存器及其各个位以通用的格式表示。小写的 "n" 表示定时器 / 计数器的序号，在此即为 2。但是在写程序时要使用精确的格式 (例如使用TCNT2来访问T/C2计数器值)。

Table 63 中定义适用于本节：

Table 63. 说明

BOTTOM	计数器计到 0x00 时即达到 BOTTOM
MAX	计数器计到 0xFF (十进制的 255) 时即达到 MAX
TOP	计数器计到计数序列的最大值时即达到 TOP。TOP 值可以为固定值 0xFF (MAX)，或是存储于寄存器 OCR2 里的数值，具体由工作模式确定

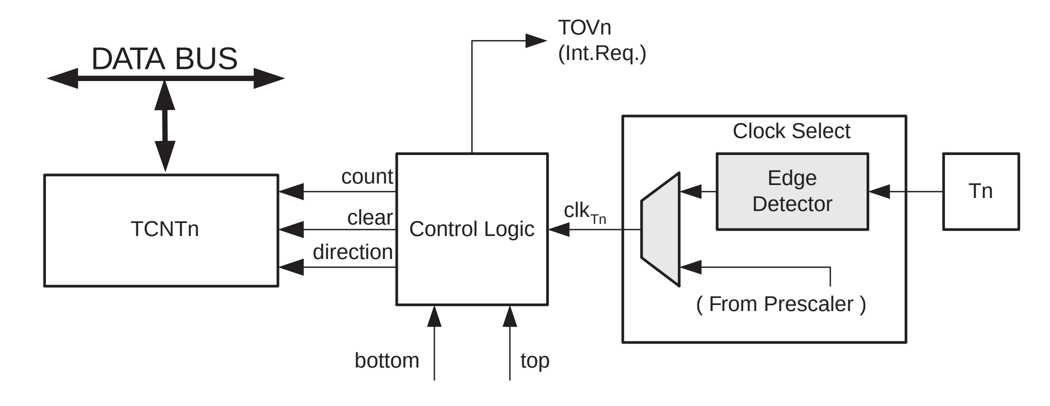
定时器 / 计数器时钟源

T/C 可以由内部同步时钟或外部异步时钟驱动。时钟源来自于 T/C 控制寄存器 TCCR2 中 CS22:0选择的连接的振荡器。详细的时钟源与预分频器的内容请参考 P 130"定时器/计数器 3、定时器 / 计数器 2 和定时器 / 计数器 1 的预分频器 " 。

计数器单元

8位T/C的主要部分为可编程的双向计数单元。Figure 62 即为计数器和它周边电路的方框图。

Figure 62. 计数器单元方框图



信号说明 (内部信号) :

count	使 TCNT2 加 1 或减 1。
direction	选择加操作或减操作。
clear	清零 TCNT2 (将所有的位清零)。
clk_{Tn}	T/C 的时钟。
top	表示 TCNT2 已经达到了最大值。
bottom	表示 TCNT2 已经达到了最小值 (0)。

根据不同的工作模式，计数器针对每一个 clk_{T2} 实现清零、加一或减一操作。clk_{T2} 可以由内部时钟源或外部时钟源产生，具体由时钟选择位 CS02:0 确定。没有选择时钟源时 (CS02:0 = 0) 定时器停止。但是不管有没有 clk_{T2} ,CPU 都可以访问 TCNT2。CPU 写操作比计数器其他操作 (清零、加减操作) 的优先级高。

计数序列由 T/C 控制寄存器 (TCCR2) 的 WGM01 和 WGM00 决定。计数器计数行为与输出比较 OC2 的波形有紧密的关系。有关计数序列和波形产生的详细信息请参考 P 135" 工作模式 " 。

T/C溢出中断标志TOV2根据WGM21:0 设定的工作模式来设置。TOV2可以用于产生CPU 中断。

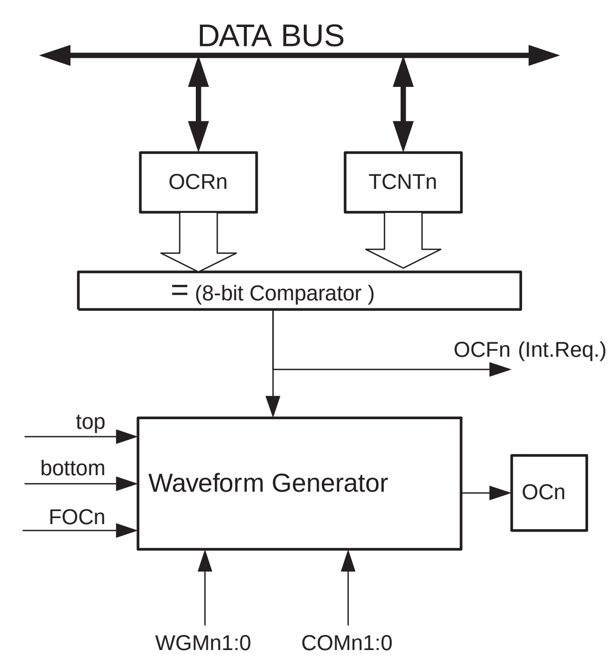




输出比较单元

8 位比较器持续对 TCNT2 和输出比较匹配寄存器 OCR2 进行比较。一旦 TCNT2 等于 OCR2，比较器就给出匹配信号。在匹配发生的下一个定时器时钟周期里输出比较标志 OCF2 置位。若 OCIE2 = 1 还将引发输出比较中断。执行中断服务程序时 OCF2 将自动清零，也可以通过软件写 ”1” 的方式进行清零。根据 WGM21:0 和 COM21:0 设定的不同工作模式，波形发生器可以利用匹配信号产生不同的波形。同时，波形发生器还利用 max 和 bottom 信号来处理极值条件下的特殊情况 (P 135 “ 工作模式 ”)。Figure 63 给出输出比较单元方框图。

Figure 63. 输出比较单元方框图



使用 PWM 模式时 OCR2 寄存器为双缓冲寄存器；而在正常工作模式和匹配时清零模式双缓冲功能是禁止的。双缓冲可以将更新 OCR2 寄存器与 top 或 bottom 时刻同步起来，从而防止产生不对称的 PWM 脉冲，消除毛刺。

访问 OCR2 寄存器看起来很复杂，其实不然。使能双缓冲功能时，CPU 访问的是 OCR2 缓冲寄存器；禁止双缓冲功能时 CPU 访问的则是 OCR2 本身。

强制输出比较

工作于非 PWM 模式时，可以对强制输出比较位 FOC2 写 ”1” 来产生比较匹配。强制比较匹配不会置位 OCF2 标志，也不会重载 / 清零定时器，但是 OC2 引脚将被更新，好象真的发生了比较匹配一样 (COM21:0 决定 OC2 是置位、清零，还是交替变化)。

写 TCNT2 将阻止比较匹配

CPU 对 TCNT2 寄存器的写操作会在下一个定时器时钟周期阻止比较匹配的发生，即使此时定时器已经停止了。这个特性可以用来将 OCR2 初始化为与 TCNT2 相同的数值而不触发中断。

使用输出比较单元

由于在任意模式下写 TCNT2 都将在下一个定时器时钟周期里阻止比较匹配，在使用输出比较时改变 TCNT2 就会有风险，不管 T/C 是否在运行。如果写入的 TCNT2 的数值等于 OCR2，比较匹配就被忽略了，造成不正确的波形发生结果。类似地，在计数器进行降序计数时不要对 TCNT2 写入 BOTTOM。

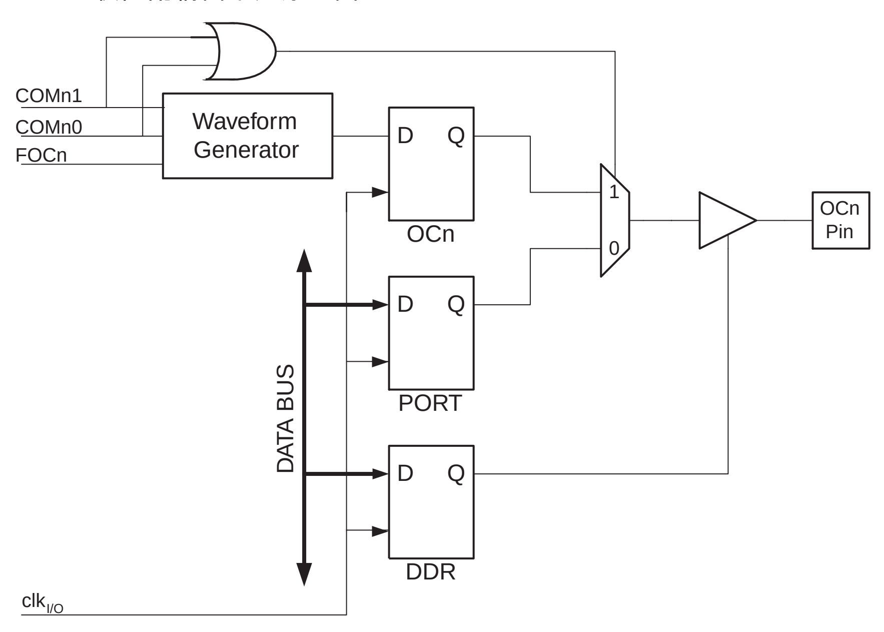
OC2 的设置应该在设置数据方向寄存器之前完成。最简单的设置 OC2 的方法是在普通模式下利用强制输出比较 FOC2。即使在改变波形发生模式时 OC2 寄存器也会一直保持它的数值。

注意 COM21:0 和比较数据都不是双缓冲的。 COM21:0 的改变将立即生效。

比较匹配输出单元

比较匹配模式控制位 COM21:0 具有双重功能。波形发生器利用 COM21:0 来确定下一次比较匹配发生时的输出比较状态 (OC2) ; COM21:0 还控制 OC2 引脚输出信号的来源。Figure 64 为受COM21:0设置影响的逻辑的简化原理图。I/O寄存器、I/O位和I/O引脚以粗体表示。图中只给出了受 COM21:0 影响的通用 I/O 端口控制寄存器 (DDR 和 PORT)。谈及 OC2 状态时指的是内部 OC2 寄存器，而不是 OC2 引脚。

Figure 64. 比较匹配输出单元原理图



只要 COM21:0 的一个或两个置位，波形发生器的输出比较功能 OC2 就会取代通用 I/O 口功能。但是 OC2 引脚的方向仍然受控于数据方向寄存器 (DDR)。在使用 OC2 功能之前首先要通过数据方向寄存器的 DDR_OC2 位将此引脚设置为输出。端口功能与波形发生器的工作模式无关。

输出比较逻辑的设计允许 OC2 状态在输出之前首先进行初始化。要注意某些 COM21:0 设置保留给了其他操作类型，详见 P 142 "8 位定时器 / 计数器寄存器说明 " 。

比较输出模式和波形产生

波形发生器利用 COM21:0 的方式在普通、 CTC 和 PWM 三种模式下有所区别。对于所有的模式，COM21:0 = 0 表明比较匹配发生时波形发生器不会操作 OC2 寄存器。非 PWM 模式的比较输出请参见 P 143Table 65 ；快速 PWM 的比较输出于 P 143Table 66 ；相位修正 PWM 的比较输出于 P 143Table 67 。

改变 COM21:0 将影响写入数据后的第一次比较匹配。对于非 PWM 模式，可以通过使用 FOC2 来强制立即产生效果。

工作模式

工作模式 - T/C 和输出比较引脚的行为 - 由波形发生模式 (WGM21:0) 及比较输出模式 (COM21:0) 的控制位决定。比较输出模式对计数序列没有影响，而波形产生模式对计数序列则有影响。 COM21:0 控制 PWM 输出是否反极性。非 PWM 模式时 COM21:0 控制输出是否应该在比较匹配发生时置位、清零，或是电平取反 (P 135 " 比较匹配输出单元 ")。

具体的时序信息请参考 P 140" 定时器 / 计数器时序图 " 的 Figure 68、Figure 69、Figure 70 与 Figure 71。

普通模式

普通模式 (WGM21:0 = 0) 为最简单的工作模式。在此模式下计数器不停地累加。计到 8 比特的最大值后 (TOP = 0xFF)，由于数值溢出计数器简单地返回到最小值 0x00 重新开始。在 TCNT2 为零的同一个定时器时钟里 T/C 溢出标志 TOV2 置位。此时 TOV2 有点象第 9 位，只是只能置位，不会清零。但由于定时器中断服务程序能够自动清零 TOV2，因此可以通过软件提高定时器的分辨率。在普通模式下没有什么需要特殊考虑的，用户可以随时写入新的计数器数值。

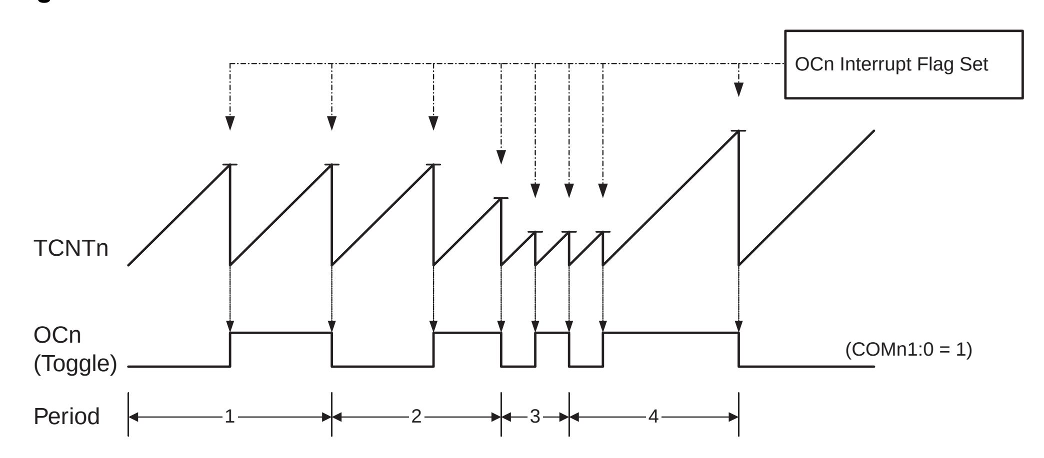
输出比较单元可以用来产生中断。但是不推荐在普通模式下利用输出比较产生波形，因为会占用太多的 CPU 时间。

CTC(比较匹配时清除定时器) 模式

在 CTC 模式 (WGM21:0 = 2) 里 OCR2 寄存器用于调节计数器的分辨率。当计数器的数值 TCNT2 等于 OCR2 时计数器清零。OCR2 定义了计数器的 TOP 值，亦即计数器的分辨率。这个模式使得用户可以很容易地控制比较匹配输出的频率，也简化了外部事件计数的操作。

CTC 模式的时序图为 Figure 65。计数器数值 TCNT2 一直累加到 TCNT2 与 OCR2 匹配，然后 TCNT0 清零。

Figure 65. CTC 模式的时序图



利用 OCF2 标志可以在计数器数值达到 TOP 即产生中断。在中断服务程序里可以更新 TOP 的数值。由于 CTC 模式没有双缓冲功能，在计数器以无预分频器或很低的预分频器工作的时候将 TOP 更改为接近 BOTTOM 的数值时要小心。如果写入 OCR2 的数值小于当前 TCNT2 的数值，计数器将丢失一次比较匹配。在下一次比较匹配发生之前，计数器不得不先计数到最大值 0xFF，然后再从 0x00 开始计数到 OCR2。

为了在 CTC 模式下得到波形输出，可以设置 OC2 在每次比较匹配发生时改变逻辑电平。这可以通过设置 COM21:0 = 1 来完成。在期望获得 OC2 输出之前，首先要将其端口设置为输出。波形发生器能够产生的最大频率为 $f_{OC0} = f_{clk_I/O}/2$ (OCR2= 0x00)。频率由如下公式确定：

$$f_{OCn} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRn)}$$

变量 N 代表预分频因子 (1、8、 64、256 或 1024)。

在普通模式下， TOV2 标志的置位发生在计数器从 MAX 变为 0x00 的定时器时钟周期。

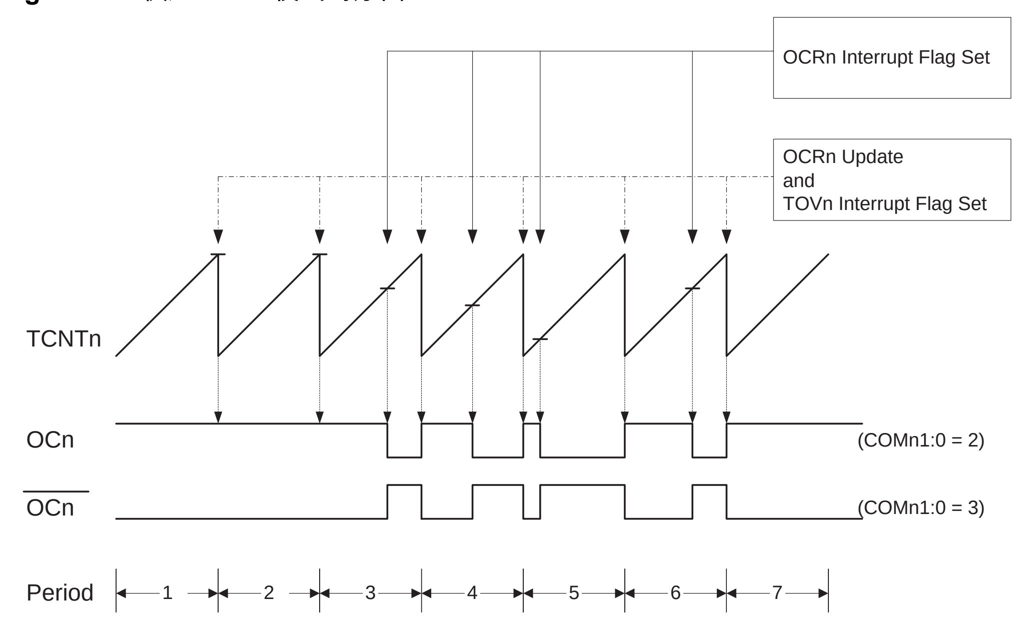
快速 PWM 模式

快速 PWM 模式 (WGM21:0 = 3) 可用来产生高频的 PWM 波形。快速 PWM 模式与其他 PWM 模式的不同之处是其单边斜坡工作方式。计数器从 BOTTOM 计到 MAX，然后立即回到 BOTTOM 重新开始。对于普通的比较输出模式，输出比较引脚 OC2 在 TCNT2 与 OCR2 匹配时清零，在 BOTTOM 时置位；对于反向比较输出模式， OC2 的动作正好相反。由于使用了单边斜坡模式，快速 PWM 模式的工作频率比使用双斜坡的相位修正 PWM 模式

高一倍。此高频操作特性使得快速 PWM 模式十分适合于功率调节，整流和 DAC 应用。高频可以减小外部元器件 (电感，电容) 的物理尺寸，从而降低系统成本。

工作于快速 PWM 模式时，计数器的数值一直增加到 MAX，然后在后面的一个时钟周期清零。具体的时序图为 Figure 66。图中柱状的 TCNT2 表示这是单边斜坡操作。方框图同时包含了普通的 PWM 输出以及方向 PWM 输出。 TCNT2 斜坡上的短水平线表示 OCR2 和 TCNT2 的比较匹配。

Figure 66. 快速 PWM 模式时序图



计时器数值达到 MAX 时 T/C 溢出标志 TOV2 置位。如果中断使能，在中断服务程序中断服务程序可以更新比较值。

工作于快速 PWM 模式时，比较单元可以在 OC2 引脚上输出 PWM 波形。设置 COM21:0 为 2 可以产生普通的 PWM 信号；为 3 则可以产生反向 PWM 波形 (参见 P 143Table 66)。要想在引脚上得到输出信号还必须将OC2的数据方向设置为输出。产生PWM波形的机理是 OC2 寄存器在 OCR2 与 TCNT2 匹配时置位 (或清零) , 以及在计数器清零 (从 MAX 变为 BOTTOM) 的那一个定时器时钟周期清零 (或置位)。

输出的 PWM 频率可以通过如下公式计算得到：

$$f_{OCnPWM} = \frac{f_{\text{clk_I/O}}}{N \cdot 256}$$

变量 N 代表分频因子 (1、8、 64、 256 或 1024)。

OCR2 寄存器为极限值时表示快速 PWM 模式的一些特殊情况。若 OCR2 等于 BOTTOM，输出为出现在第 MAX+1 个定时器时钟周期的窄脉冲； OCR2 为 MAX 时 , 根据 COM21:0 的设定，输出恒为高电平或低电平。

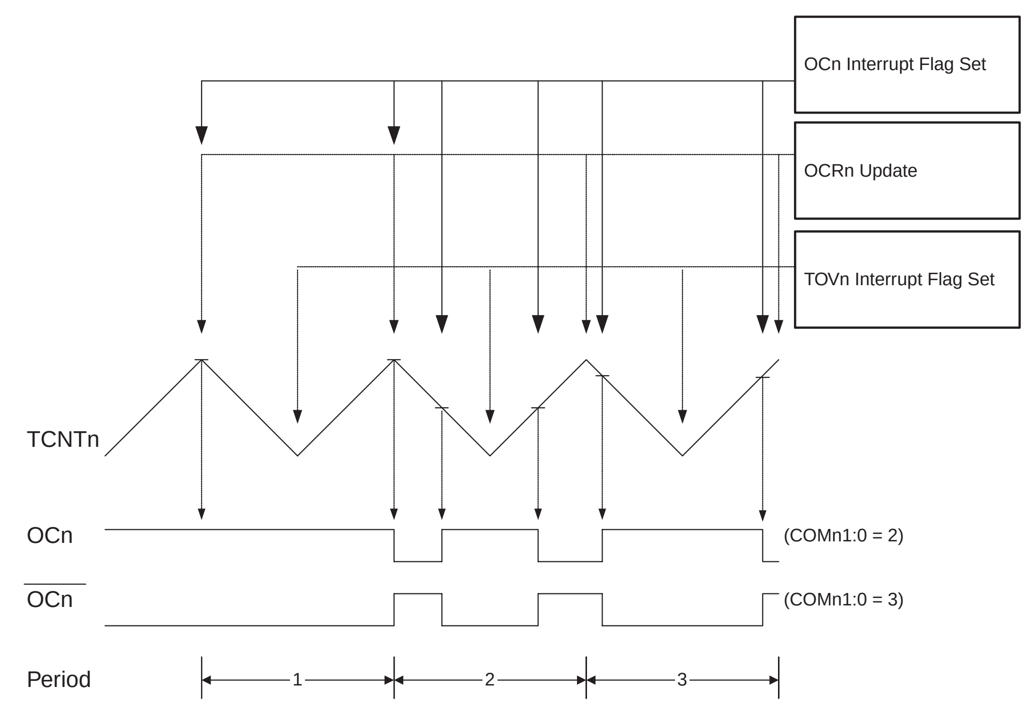
通过设定 OC2 在比较匹配时进行逻辑电平取反 (COM21:0 = 1)，可以得到占空比为 50% 的周期信号。OCR2 为 0 时信号有最高频率 $f_{oc0} = f_{clk_I/O}/2$。这个特性类似于 CTC 模式下的 OC2 取反操作，不同之处在于快速 PWM 模式具有双缓冲。

相位修正 PWM 模式

相位修正 PWM 模式 (WGM21:0 = 1) 为用户提供了一个获得高精度相位修正 PWM 波形的方法。此模式基于双斜坡操作。计时器重复地从 BOTTOM 计到 MAX，然后又从 MAX 倒退回到BOTTOM。在一般的比较输出模式下，当计时器往MAX计数时若发生了TCNT2 于OCR2的匹配，OC2将清零为低电平；而在计时器往BOTTOM计数时若发生了TCNT2 于 OCR2 的匹配， OC2 将置位为高电平。工作于反向输出比较时则正好相反。与单斜坡操作相比，双斜坡操作可获得的最大频率要小。但由于其对称的特性，十分适合于电机控制。

相位修正 PWM 模式的 PWM 精度固定为 8 比特。计时器不断地累加直到 MAX，然后开始减计数。在一个定时器时钟周期里 TCNT2 的值等于 MAX。时序图可参见 Figure 67。图中 TCNT2 的数值用柱状图表示，以说明双斜坡操作。本图同时说明了普通 PWM 的输出和反向 PWM 的输出。 TCNT2 斜坡上的小横条表示 OCR2 和 TCNT2 的比较匹配。

Figure 67. 相位修正 PWM 模式的时序图



当计时器达到 BOTTOM 时 T/C 溢出标志位 TOV2 置位。此标志位可用来产生中断。

工作于相位修正 PWM 模式时，比较单元可以在 OC2 引脚产生 PWM 波形：将 COM21:0 设置为2产生普通相位的PWM，设置COM21:0为3产生反向PWM信号 (参见P 143Table 67)。要想在引脚上得到输出信号还必须将OC2的数据方向设置为输出。OCR2和 TCNT2 比较匹配发生时 OC2 寄存器将产生相应的清零或置位操作，从而产生 PWM 波形。工作于相位修正模式时 PWM 频率可由下式公式获得：

$$f_{OCnPCPWM} = \frac{f_{clk_I/O}}{N \cdot 510}$$

变量 N 表示预分频因子 (1、8、64、 256 或 1024)。

OCR2寄存器处于极值代表了相位修正PWM模式的一些特殊情况。在普通PWM模式下，若 OCR2 等于 BOTTOM，输出一直保持为低电平；若 OCR2 等于 MAX，则输出保持为高电平。反向 PWM 模式则正好相反。

Figure 67 中的第二个时钟周期中，即使没有比较匹配，OCn 电平也会由高变低。这样保证关于 BOTTOM 对称。在以下两种情况下没有比较匹配时电平发生变化：

- 如 Figure 67 中 OCR0A 值由 MAX 改变。降序比较匹配时 , 当 OCR0A 值为 MAX , OCn 引脚的值也一样；为保证关于 BOTTOM 对称，在升序比较匹配时， OCn 值为 MAX 时电平变化。
- 定时器开始计数的值大于 OCR0A 中的值，因此少一次比较匹配，且在达到上限时 OCn 改变。



定时器 / 计数器时序图

下面图中给出的 T/C 为同步电路，因此其时钟 clk_{T2} 可以表示为时钟使能信号。图中还说明了中断标志设置的时间。 Figure 68 包含了 T/C 的基本时序。图中给出了除相位修正 PWM 模式的接近 MAX 值的计数序列。

Figure 68. T/C 时序图，无预分频器

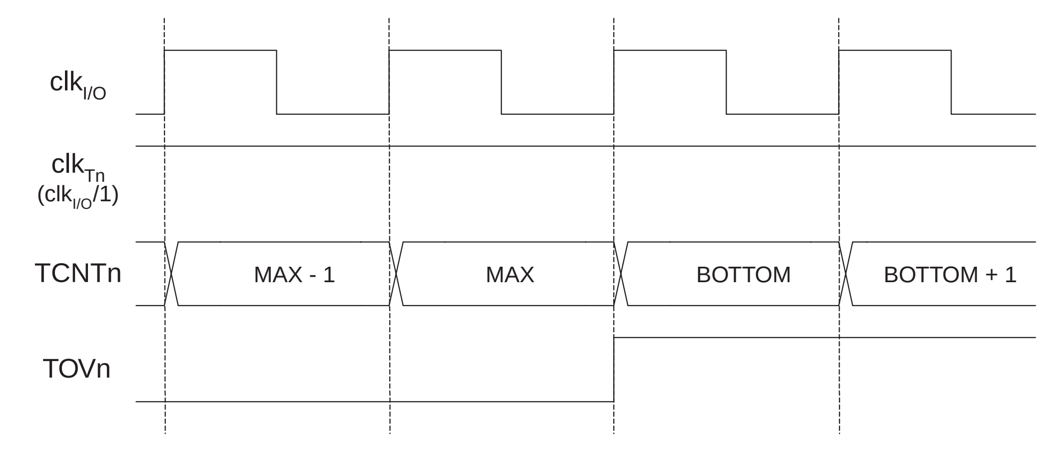


Figure 69 给出相同的定时数据，但预分频器使能。

Figure 69. T/C 时序图，预分频器为 $f_{clk_I/O}/8$

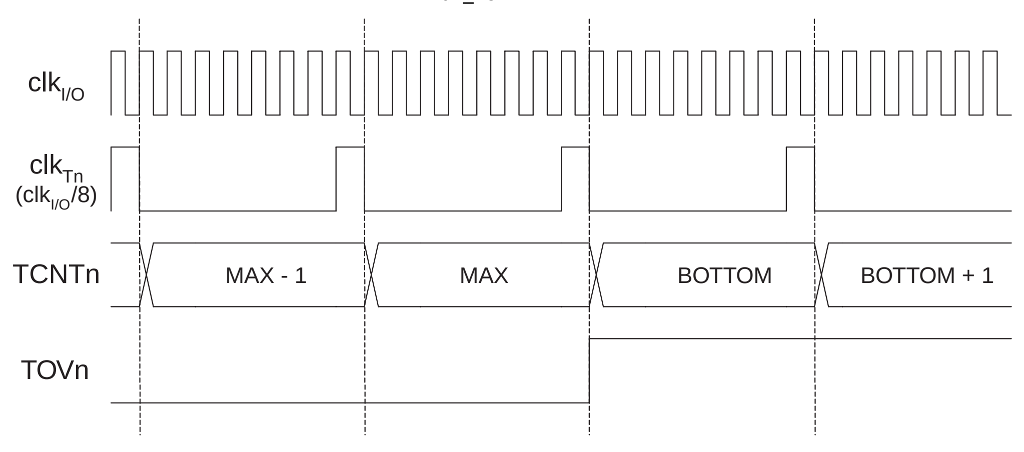
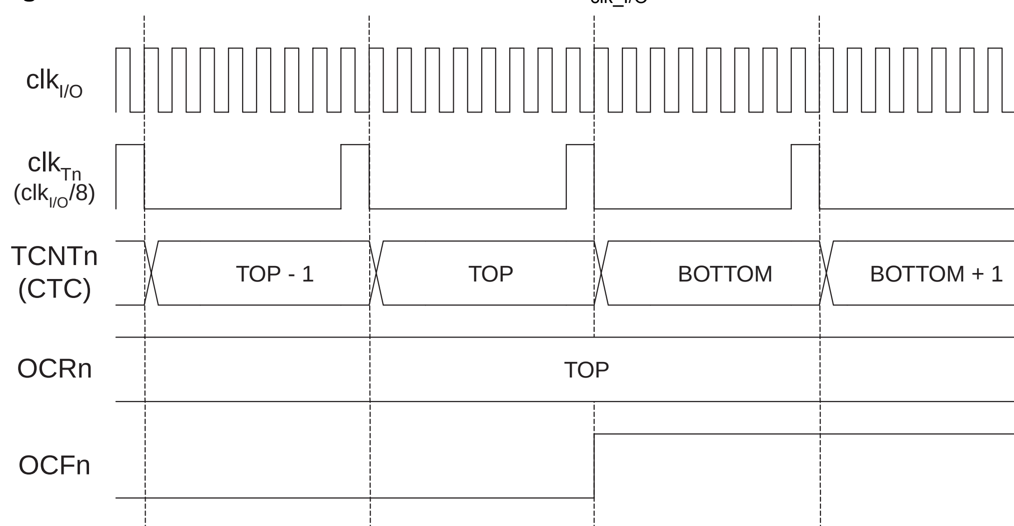


Figure 70 给出了各种模式下 (除了 CTC 模式)OCF2 的置位情况。

Figure 70. T/C 时序图， OCF2 置位，预分频器为 $f_{clk_I/O}/8$

Figure 71 给出了 CTC 模式下 OCF2 置位和 TCNT2 清除的情况。

Figure 71. T/C 时序图， CTC 模式，预分频器为 $f_{clk_I/O}/8$



8 位定时器 / 计数器寄存器说明

定时器 / 计数器控制寄存器 - TCCR2

Bit	7	6	5	4	3	2	1	0	
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	TCCR2
读 / 写	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC2: 强制输出比较**

FOC2 仅在 WGM20 位指明非 PWM 模式时才有效。但是 ,为了保证与未来器件的兼容性，使用 PWM 时，写 TCCR2 要对其清零。写 1 后，波形发生器将立即进行比较操作。比较匹配输出引脚 OC2 将按照 COM21:0 的设置输出相应的电平。要注意 FOC2 类似一个锁存信号，真正对强制输出比较起作用的是 COM21:0 的设置。

FOC2 不会引发任何中断 , 也不会在使用 OCR2 作为 TOP 的 CTC 模式下对定时器进行清零。

读 FOC2 的返回值永远为 0。

- **Bit 6, 3 – WGM21:0: 波形产生模式**

这几位控制计数器的计数序列，计数器最大值 TOP 的来源，以及产生何种波形。 T/C 支持的模式有：普通模式，比较匹配发生时清除计数器模式 (CTC)，以及两种 PWM 模式，详见 Table 64 与 P 135“ 工作模式 ” 。

Table 64. 波形产生模式的位定义

模式	WGM21 (CTC2)	WGM20 (PWM2)	T/C 的工作模式	TOP	OCR2 的更新时间	TOV2 的置位时刻
0	0	0	普通	0xFF	立即更新	MAX
1	0	1	相位修正 PWM	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR2	立即更新	MAX
3	1	1	快速 PWM	0xFF	TOP	MAX

Note: 位定义 CTC2 和 PWM2 已经不再使用了 , 要使用 WGM21:0。但是功能和位置与以前版本兼容。

- **Bit 5:4 – COM21:0: 比较匹配输出模式**

这些位决定了比较匹配发生时输出引脚 OC2 的电平。如果 COM21:0 中的一位或全部都置位， OC2 以比较匹配输出的方式进行工作。同时其方向控制位要设置为 1 以使能输出驱动。

当 OC2 连接到物理引脚上时， COM21:0 的功能依赖于 WGM21:0 的设置。 Table 65 给出了当 WGM21:0 设置为普通模式或 CTC 模式时 COM21:0 的功能。

Table 65. 比较输出模式，非 PWM 模式

COM21	COM20	说明
0	0	正常的端口操作， OC2 未连接
0	1	比较匹配发生时 OC2 取反
1	0	比较匹配发生时 OC2 清零
1	1	比较匹配发生时 OC2 置位

Table 66 给出了当 WGM21:0 设置为快速 PWM 模式时 COM21:0 的功能。

Table 66. 比较输出模式，快速 PWM 模式 (1)

COM21	COM20	说明
0	0	正常的端口操作， OC2 未连接
0	1	保留
1	0	比较匹配发生时 OC2 清零 ，计数到 TOP 时 OC2 置位
1	1	比较匹配发生时 OC2 置位 ，计数到 TOP 时 OC2 清零

Note: 1. 一个特殊情况是 OCR2 等于 TOP，且 COM21 置位。此时比较匹配将被忽略，而计数到 TOP 时的动作继续有效。详细信息请参见 P 136" 快速 PWM 模式 " 。

Table 67 给出了当 WGM21:0 设置为相位修正 PWM 模式时 COM21:0 的功能。

Table 67. 比较输出模式，相位修正 PWM 模式 (1)

COM21	COM20	说明
0	0	正常的端口操作， OC2 未连接
0	1	保留
1	0	在升序计数时发生比较匹配将清零 OC2 ；降序计数时发生比较匹配将置位 OC2
1	1	在升序计数时发生比较匹配将置位 OC2 ；降序计数时发生比较匹配将清零 OC2

Note: 1. 一个特殊情况是 OCR2 等于 TOP，且 COM21 置位。此时比较匹配将被忽略，而计数到 TOP 时的动作继续有效。详细信息请参见 P 138" 相位修正 PWM 模式 " 。

- **Bit 2:0 – CS22:0: 时钟选择**

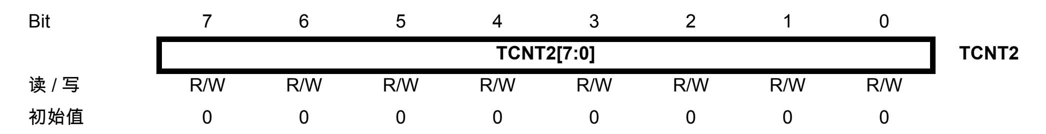
这三位时钟选择位用于选择 T/C 的时钟源。

Table 68. 时钟选择位定义

CS22	CS21	CS20	说明
0	0	0	无时钟， T/C 不工作
0	0	1	$clk_{I/O}$/(没有预分频)
0	1	0	$clk_{I/O}$/8 (来自预分频器)
0	1	1	$clk_{I/O}$/64 (来自预分频器)
1	0	0	$clk_{I/O}$/256 (来自预分频器)
1	0	1	$clk_{I/O}$/1024 (来自预分频器)
1	1	0	从 T2 引脚的外部时钟源。时钟为下降沿
1	1	1	从 T2 引脚的外部时钟源。时钟为上升沿

如果 T/C2 使用外部引脚模式 , 即使 T2 引脚作为输出 , 其上的变化也作为计数器的时钟。该特性允许软件控制计数。

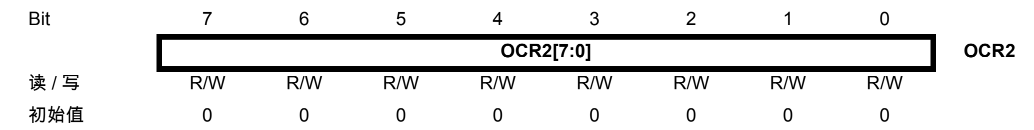
定时器 / 计数器寄存器 - TCNT2



Bit	7	6	5	4	3	2	1	0	
	TCNT2[7:0]								TCNT2
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

通过 T/C 寄存器可以直接对计数器的 8 位数据进行读写访问。对 TCNT2 寄存器的写访问将在下一个时钟阻止比较匹配。在计数器运行的过程中修改 TCNT2 的数值有可能丢失一次 TCNT2 和 OCR2 的比较匹配。

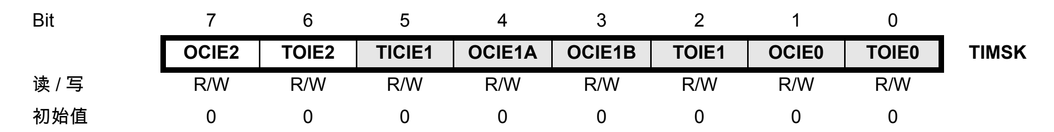
输出比较寄存器 - OCR2



Bit	7	6	5	4	3	2	1	0	
	OCR2[7:0]								OCR2
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

输出比较寄存器包含一个 8 位的数据 , 不间断地与计数器数值 TCNT2 进行比较。匹配事件可以用来产生输出比较中断 , 或者用来在 OC2 引脚上产生波形。

定时器 / 计数器中断屏蔽寄存器 - TIMSK



Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – OCIE2:T/C2 输出比较匹配中断使能**

当 OCIE2 和状态寄存器的全局中断使能位 I 都为 "1" 时 , T/C2 的输出比较匹配中断使能。当 T/C2 的比较匹配发生 , 即 TIFR 中的 OCF2 置位时 , 中断服务程序得以执行。

- **Bit 6 – TOIE2:T/C2 溢出中断使能**

当 TOIE2 和状态寄存器的全局中断使能位 I 都为 "1" 时 , T/C2 的溢出中断使能。当 T/C2 发生溢出 , 即 TIFR 中的 TOV2 位置位时 , 中断服务程序得以执行。

定时器 / 计数器中断标志寄存器 - TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – OCF2: 输出比较标志 2**

当 T/C2 与 OCR2(输出比较寄存器 2) 的值匹配时 , OCF2 置位。此位在中断服务程序里硬件清零 , 也可以通过对其写 1 来清零。当 SREG 中的位 I、OCIE2 和 OCF2 都置位时 , 中断服务程序得到执行。

- **Bit 6 – TOV2:T/C2 溢出标志**

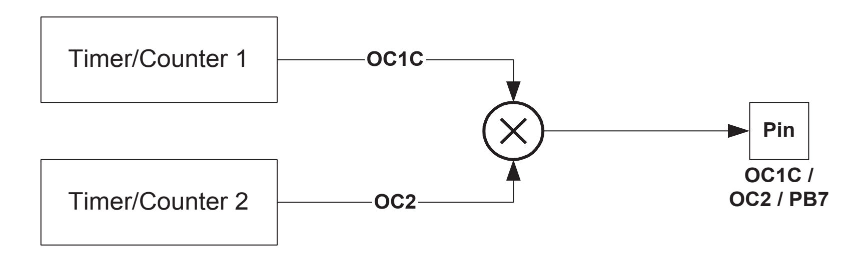
当 T/C2 溢出时 , TOV2 置位。执行相应的中断服务程序时此位硬件清零。此外 , TOV2 也可以通过写 1 来清零。当 SREG 中的位 I、 TOIE2 和 TOV2 都置位时 , 中断服务程序得到执行。在 PWM 模式中 , 当 T/C2 在 $00 改变记数方向时 , TOV2 置位。

输出比较调制器 (OCM1C2)

综述

输出比较调制器 (OCM) 可产生由载波频率调制的波形。调制器使用 16 位 T/C1 的输出比较单元及 8 位 T/C2 的输出比较单元作为其输出。更多的有关 T/C 的内容请见 P 100"16 位定时器 / 计数器 (定时器 / 计数器 1 和定时器 / 计数器 3)" 及 P 132" 具有 PWM 功能的 8 位定时器 / 计数器 2" 。注意该特性在 ATmega103 兼容模式下无效。

Figure 72. 输出比较调制器框图



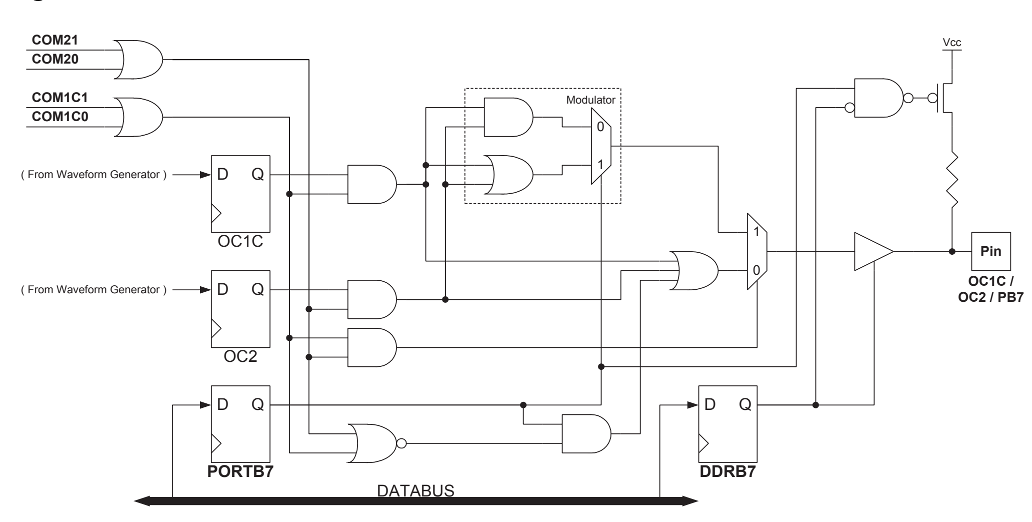
当该调制器使能，两个输出比较通路会如 Figure 72 所示，被同时调制。

说明

输出比较单元 1C 与 2 共用端口 PB7 作为输出引脚。当其中之一使能 (即当 COMnx1:0 不为 0) 输出比较单元 (OC1C 与 OC2) 覆盖通用 PORTB7 寄存器功能。当 OC1C 与 OC2 同时使能，调制器自动使能。

Figure 73 给出调制器的示意图。图中包括 T/C 单元及端口 B 引脚 7 输出驱动电路。

Figure 73. 输出比较调制器

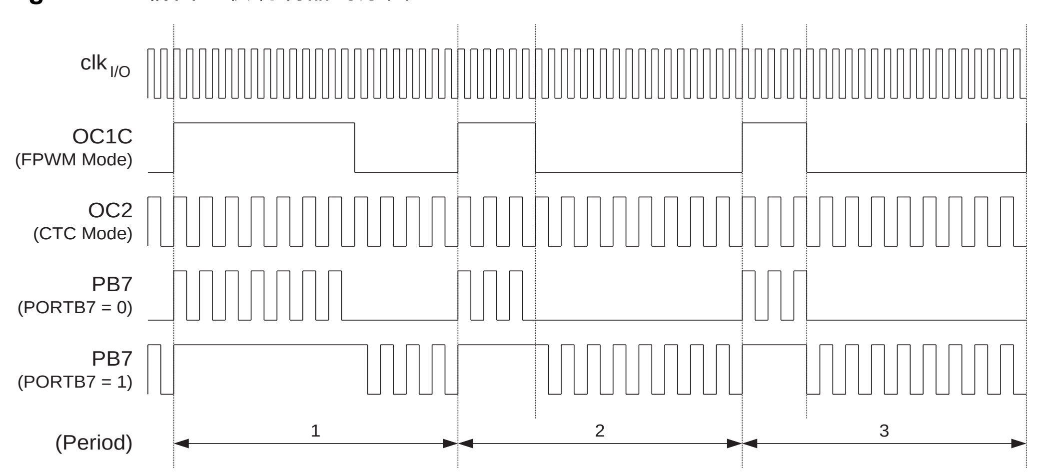


当调制器使能后 , 调制类型 (逻辑 AND 或 OR) 由 PORTB7 寄存器决定。注意由 COMnx1:0 设置的 DDRB7 控制端口方向。

时序例子

Figure 74给出调制器的运行状况。本例中设置T/C1工作在快速 PWM模式下 T/C2 使用由比较输出模式 (COMnx1:0 = 1) 的 CTC 波形模式。

Figure 74. 输出比较调制器时序图



本例中， T/C2 提供载波，调制信号由 T/C1 的输出比较单元 C 产生。

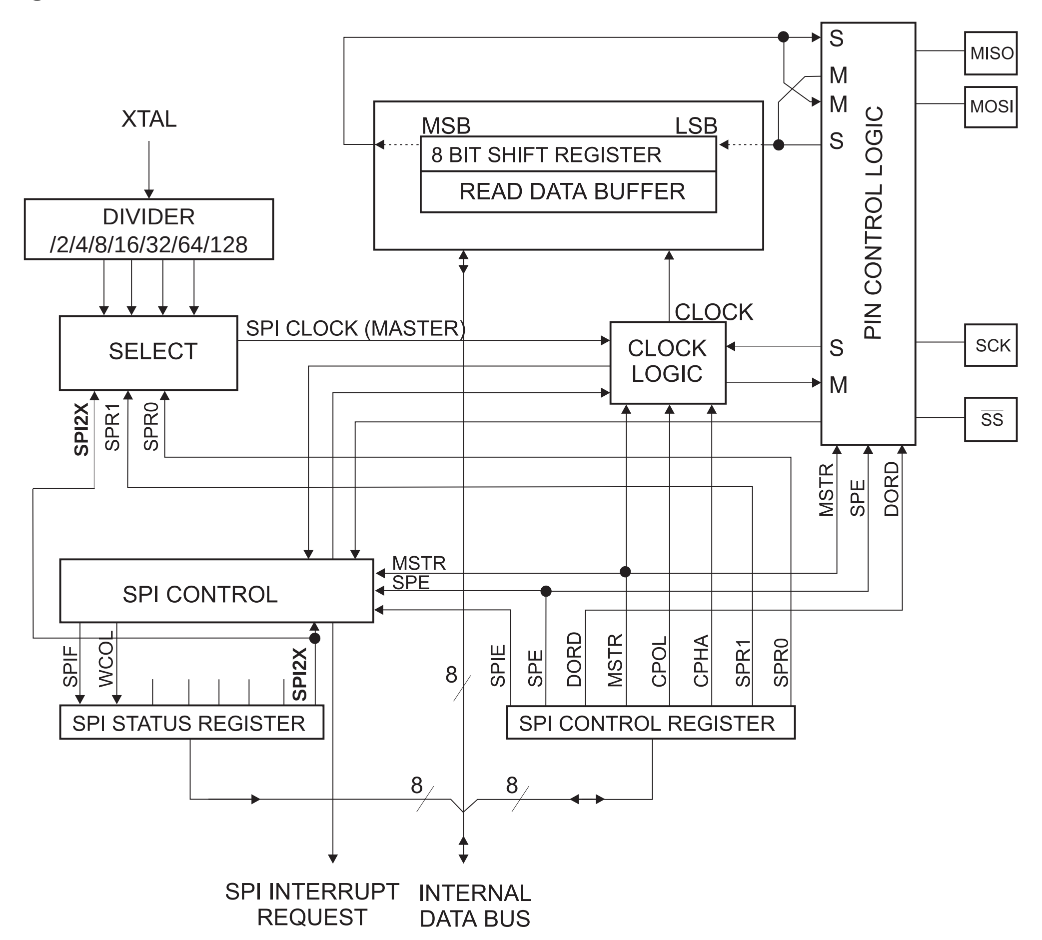
经过调制 PWM 信号 (OC1C) 的分辨率降低。缩减因子等于一个载波周期 (OC2) 中系统时钟循环数。在本例中分辨率降为原来的一半。下降原因在Figure 74中示出，即当PORTB7 为 0 时 PB7 输出的第二和第三周期。周期 2 高电平时间为周期 3 高电平时间的一倍，而 PB7 输出的结果相同。

串行通讯接口 - SPI

串行外设接口 SPI 允许 ATmega128 和外设之间进行高速的同步数据传输。 ATmega128 SPI 的特点如下：

- **全双工， 3 线同步数据传输**
- **主机或从机操作**
- **LSB 首先发送或 MSB 首先发送**
- **7 种可编程的比特率**
- **传输结束中断**
- **写碰撞标志检测**
- **可以从闲置模式唤醒**
- **作为主机时具有双速模式 (CK/2)**

Figure 75. SPI 方框图



Note: SPI 的引脚排列请参见 P 2Figure 1 和 P 69Table 30 。

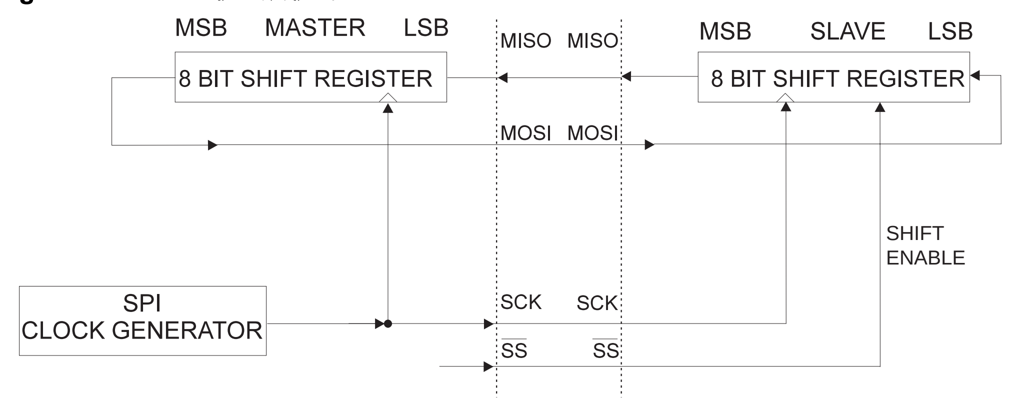
主机和从机之间的 SPI 连接如 Figure 76 所示。系统包括两个移位寄存器和一个主机时钟发生器。通过将需要的从机的 $\overline{SS}$ 引脚拉低，主机启动一次通讯过程。主机和从机将需要发送的数据放入相应的移位寄存器。主机在 SCK 引脚上产生时钟脉冲以交换数据。主机的数据从主机的 MOSI 移出，从从机的 MOSI 移入；从机的数据从从机的 MISO 移出，从主机的 MISO 移入。主机通过将从机的 $\overline{SS}$ 拉高实现与从机的同步。

配置为 SPI 主机时， SPI 接口不控制 $\overline{SS}$ 引脚，必须由用户软件来处理。对 SPI 数据寄存器写入数据即启动 SPI 时钟，将 8 比特的数据移入从机。传输结束后 SPI 时钟停止，传输结束标志 SPIF 置位。如果此时 SPCR 寄存器的 SPI 中断使能位 SPIE 置位，中断就会发生。主机可以继续往 SPDR 写入数据以移位到从机中去，或者是将从机的 $\overline{SS}$ 拉高以说明数据包发送完成。最后进来的数据将一直保存于缓冲寄存器里。



配置为从机时，只要 $\overline{SS}$ 为高，SPI 接口将一直保持睡眠状态，并保持 MISO 为三态。在这个状态下软件可以更新 SPI 数据寄存器 SPDR 的内容。一个字节完全移出之后，传输结束标志 SPIF 置位。如果此时 SPCR 寄存器的 SPI 中断使能位 SPIE 置位，中断就会发生。在读取移入的数据之前从机可以继续往 SPDR 写入数据。最后进来的数据将一直保存于缓冲寄存器里。

Figure 76. SPI 主机 - 从机的互连



SPI 系统的发送方向只有一个缓冲器，而在接收方向有两个缓冲器。也就是说，在发送时一定要等到移位过程全部结束后才能对 SPI 数据寄存器执行写操作。而在接收数据时，需要在下一个字符移位过程结束前通过访问SPI 数据寄存器读取当前接收到的字符。否则第一个字节将丢失。

工作于 SPI 从机模式时，控制逻辑对 SCK 引脚的输入信号进行采样。为了保证对时钟信号的正确采样， SPI 时钟不能超过 $f_{osc}/4$。

SPI 使能后 ,MOSI、MISO、SCK 和 $\overline{SS}$ 引脚的数据方向将按照 Table 69 所示自动进行配置。更多自动重载信息请参考 P 66" 端口的第二功能 " 。

Table 69. SPI 引脚重载 (1)

引脚	方向， SPI 主机	方向， SPI 从机
MOSI	用户定义	输入
MISO	输入	用户定义
SCK	用户定义	输入
$\overline{SS}$	用户定义	输入

Note: 1. 请参考 P 69" 端口 B 的第二功能 " 以了解如何定义由用户定义的 SPI 引脚。

下面的例子说明如何将 SPI 初始化为主机，以及如何进行简单的数据发送。例子中 DDR_SPI必须由实际的数据方向寄存器代替；DD_MOSI、DD_MISO和DD_SCK必须由

实际的数据方向代替。比如说， MOSI 为 PB5 引脚，则 DD_MOSI 要用 DDB5 取代，DDR_SPI 则用 DDRB 取代。

汇编代码例子 [1]

```
SPI_MasterInit:
  ; 设置 MOSI 和 SCK 为输出，其他为输入
  ldi   r17,(1<<DD_MOSI)|(1<<DD_SCK)
  out   DDR_SPI,r17
  ; 使能 SPI 主机模式，设置时钟速率为 fck/16
  ldi   r17,(1<<SPE)|(1<<MSTR)|(1<<SPR0)
  out   SPCR,r17
  ret

SPI_MasterTransmit:
  ; 启动数据传输 (r16)
  out   SPDR,r16
Wait_Transmit:
  ; 等待传输结束
  sbis  SPSR,SPIF
  rjmp  Wait_Transmit
  ret
```

C 代码例子 [1]

```
void SPI_MasterInit(void)
{
  /* 设置 MOSI 和 SCK 为输出，其他为输入 */
  DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
  /* 使能 SPI 主机模式，设置时钟速率为 fck/16 */
  SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}

void SPI_MasterTransmit(char cData)
{
  /* 启动数据传输 */
  SPDR = cData;
  /* 等待传输结束 */
  while(!(SPSR & (1<<SPIF)))
    ;
}
```

Note: 1. 例子假定已经包含了正确的头文件。

下面的例子说明如何将 SPI 初始化为从机，以及如何进行简单的数据接收。

汇编代码例子 [1]

```
SPI_SlaveInit:
  ; 设置 MISO 为输出，其他为输入
  ldi   r17,(1<<DD_MISO)
  out   DDR_SPI,r17
  ; 使能 SPI
  ldi   r17,(1<<SPE)
  out   SPCR,r17
  ret

SPI_SlaveReceive:
  ; 等待接收结束
  sbis  SPSR,SPIF
  rjmp  SPI_SlaveReceive
  ; 读取接收到的数据，然后返回
  in    r16,SPDR
  ret
```

C 代码例子 [1]

```
void SPI_SlaveInit(void)
{
  /* 设置 MISO 为输出，其他为输入 */
  DDR_SPI = (1<<DD_MISO);
  /* 使能 SPI */
  SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
  /* 等待接收结束 */
  while(!(SPSR & (1<<SPIF)))
    ;
  /* 返回数据 */
  return SPDR;
}
```

Note: 1. 例子假定已经包含了正确的头文件。

$\overline{SS}$ 引脚的功能

从机模式

当 SPI 配置为主机时，从机选择引脚 $\overline{SS}$ 总是输入。$\overline{SS}$ 为低将激活 SPI 接口， MISO 成为输出 (用户必须进行相应的配置) 引脚，其他引脚成为输入引脚。当 $\overline{SS}$ 为高时所有的引脚成为输入， SPI 接口复位，不再接收数据。

$\overline{SS}$ 引脚对于数据包/字节的同步非常有用，可以使从机和主机同步。当 $\overline{SS}$ 拉高时SPI从机立即复位接收和发送逻辑，移位寄存器里的数据有可能是不完整的数据。

主机模式

当 SPI 配置为主机时 （ MSTR 的 SPCR 置位)，用户可以决定 $\overline{SS}$ 引脚的方向。

若 $\overline{SS}$ 配置为输出，则此引脚可以用作普通的 I/O 口而不影响 SPI 系统。典型应用是用来驱动从机的 $\overline{SS}$ 引脚。

如果 $\overline{SS}$ 配置为输入，必须保持为高以保证 SPI 的正常工作。若系统配置为主机， $\overline{SS}$ 为输入，但被外设拉低，则 SPI 系统会将此低电平解释为有一个外部主机将自己选择为从机。为了防止总线冲突， SPI 系统遵循以下规则：

1. 如果 SPCR 的 MSTR 位为 '0'，则 SPI 成为从机， MOSI 和 SCK 变为输入。
2. 如果 SPSR 的 SPIF 置位，且 SPI 中断和全局中断开放，则中断例程将得到执行。

因此，使用中断方式处理 SPI 主机的数据传输，并且存在 $\overline{SS}$ 被拉低的可能性时，中断例程应该检查 MSTR 是否为 '1'。若被清零，用户必须将其置位，以重新使能 SPI 主机模式。

SPI 控制寄存器 - SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIE: SPI 中断使能**

置位后，只要 SPSR 寄存器的 SPIF 和 SREG 寄存器的全局中断使能位置位，就会引发 SPI 中断。

- **Bit 6 – SPE: SPI 使能**

SPE 置位将使能 SPI。

- **Bit 5 – DORD: 数据次序**

DORD 置位时数据的 LSB 首先发送；否则数据的 MSB 首先发送。

- **Bit 4 – MSTR: 主 / 从选择**

MSTR 置位时选择主机模式，否则为从机。如果 MSTR 为 '1'，$\overline{SS}$ 配置为输入，但被拉低，则 MSTR 被清零，寄存器 SPSR 的 SPIF 置位。用户必须重新设置 MSTR 进入主机模式。

- **Bit 3 – CPOL: 时钟极性**

CPOL为高表示空闲时SCK为高电平；否则，表示空闲时SCK为低电平。请参考Figure 77 和 Figure 78。

Table 70. CPOL 的功能

CPOL	起始沿	结束沿
0	上升沿	下降沿
1	下降沿	上升沿

- **Bit 2 – CPHA: 时钟相位**



CPHA 决定数据是在 SCK 的起始沿采样还是在 SCK 的结束沿采样。请参考 Figure 77 和 Figure 78。

Table 71. CPHA 的功能

CPHA	起始沿	结束沿
0	采样	设置
1	设置	采样

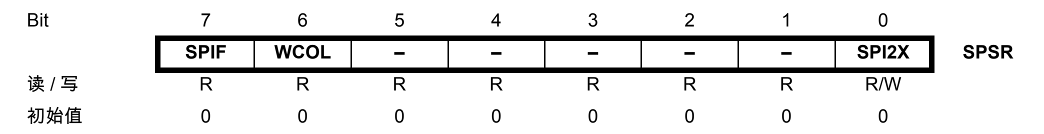
- **Bits 1, 0 – SPR1, SPR0: SPI 时钟速率选择 1 和 0**

确定主机的 SCK 速率。SPR1 和 SPR0 对从机没有影响。 SCK 和振荡器的时钟频率 f_{osc} 关系如下表所示：

Table 72. SCK 和振荡器频率的关系

SPI2X	SPR1	SPR0	SCK 频率
0	0	0	f_{osc} /4
0	0	1	f_{osc} /16
0	1	0	f_{osc} /64
0	1	1	f_{osc} /128
1	0	0	f_{osc} /2
1	0	1	f_{osc} /8
1	1	0	f_{osc} /32
1	1	1	f_{osc} /64

SPI 状态寄存器 - SPSR



Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
读 / 写	R	R	R	R	R	R	R	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – SPIF: SPI 中断标志**

串行发送结束后，SPIF 置位。若此时寄存器 SPCR 的 SPIE 和全局中断使能位置位，SPI 中断即产生。如果 SPI 为主机，$\overline{SS}$ 配置为输入，且被拉低，SPIF 也将置位。进入中断例程后 SPIF 自动清零。或者可以通过先读 SPSR，紧接着访问 SPDR 来对 SPIF 清零。

- **Bit 6 – WCOL: 写冲突标志**

在发送当中对 SPI 数据寄存器 SPDR 写数据将置位 WCOL。WCOL 可以通过先读 SPSR，紧接着访问 SPDR 来清零。

- **Bit 5..1 – Res: 保留**

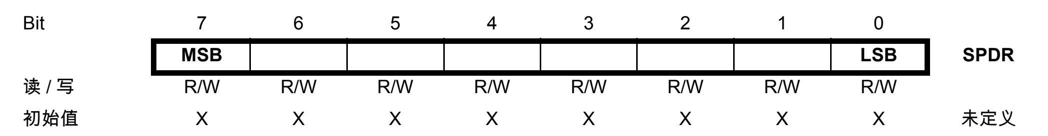
保留位，读操作返回值为零。

- **Bit 0 – SPI2X:SPI 倍速**

置位后 SPI 的速度加倍。若为主机，则 SCK 频率可达 CPU 频率的一半。若为从机，只能保证 $f_{osc}/4$。

ATmega128 的 SPI 接口同时还用来实现程序和 EEPROM 的下载和上载。请参见 SPI 串行编程和校验。

SPI 数据寄存器 - SPDR



Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	SPDR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	X	X	X	X	X	X	X	X	未定义

SPI 数据寄存器为读 / 写寄存器，用来在寄存器文件和 SPI 移位寄存器之间传输数据。写寄存器将启动数据传输，读寄存器将读取寄存器的接收缓冲器。

数据模式

SCK 的相位、极性与数据间有 4 种组合。CPHA 和 CPOL 控制组合的方式。SPI 数据传输格式见 Figure 77 和 Figure 78。数据每一位的移出和移入发生于 SCK 不同的信号跳变沿，以保证有足够的时间使数据稳定。这个过程在 Table 70 和 Table 71 有清楚的说明。

Table 73. CPOL 与 CPHA 功能

	起始沿	结束沿	SPI 模式
CPOL = 0, CPHA = 0	采样 (上升沿)	设置 (下降沿)	0
CPOL = 0, CPHA = 1	设置 (上升沿)	采样 (下降沿)	1
CPOL = 1, CPHA = 0	采样 (下降沿)	设置 (上升沿)	2
CPOL = 1, CPHA = 1	设置 (下降沿)	采样 (上升沿)	3

Figure 77. CPHA = 0 时 SPI 的传输格式

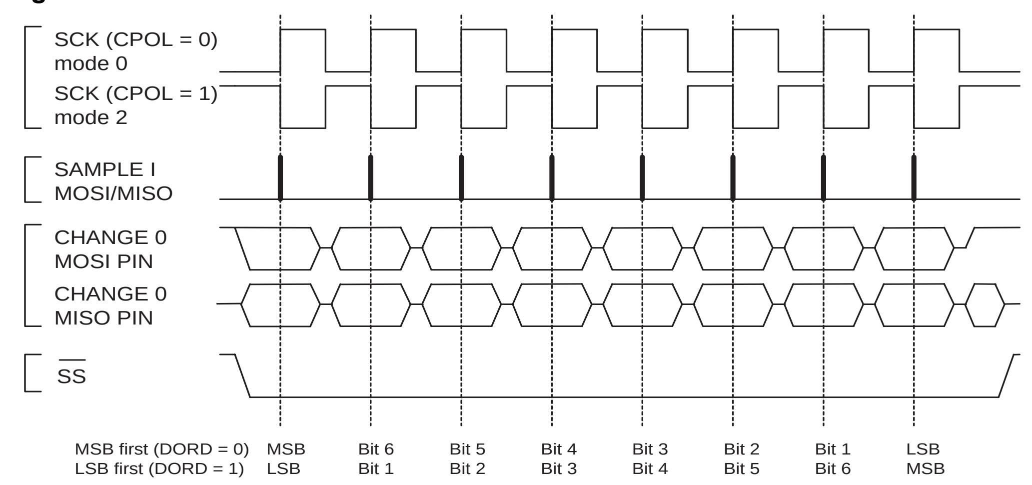
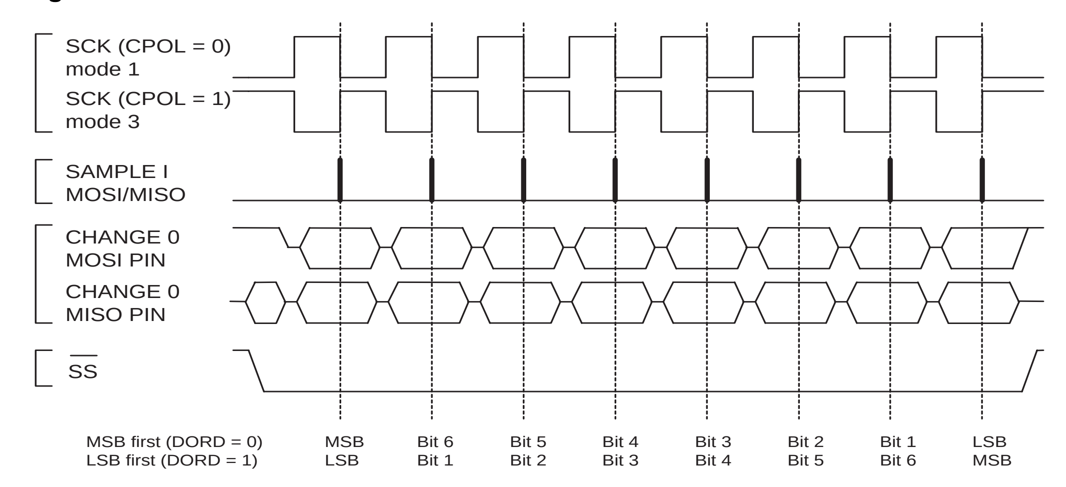


Figure 78. CPHA = 1 时 SPI 的传输格式



USART

通用同步和异步串行接收器和转发器 (USART) 是一个高度灵活的串行通讯设备。主要特点为 :

- **全双工操作 (独立的串行接收和发送寄存器)**
- **异步或同步操作**
- **主机或从机提供时钟的同步操作**
- **高精度的波特率发生器**
- **支持 5, 6, 7, 8, 或 9 个数据位和 1 个或 2 个停止位**
- **硬件支持的奇偶校验操作**
- **数据过速检测**
- **帧错误检测**
- **噪声滤波，包括错误的起始位检测，以及数字低通滤波器**
- **三个独立的中断：发送结束中断 , 发送数据寄存器空中断，以及接收结束中断**
- **多处理器通讯模式**
- **倍速异步通讯模式**

双 USART

ATmega128 具有两个 USART ,USART0 和 USART1。两个 USART 的功能在下面说明。USART0 和 USART1 具有不同的 I/O 寄存器 , 如 P 342“ 寄存器概述 ” 所述。在 ATmega103 兼容模式下 USART1 是不可见的， UBRR0H 和 UCRS0C 寄存器也是如此。也就是说，在 ATmega103 兼容模式下 ATmega128 只支持一个异步工作的 USART0。

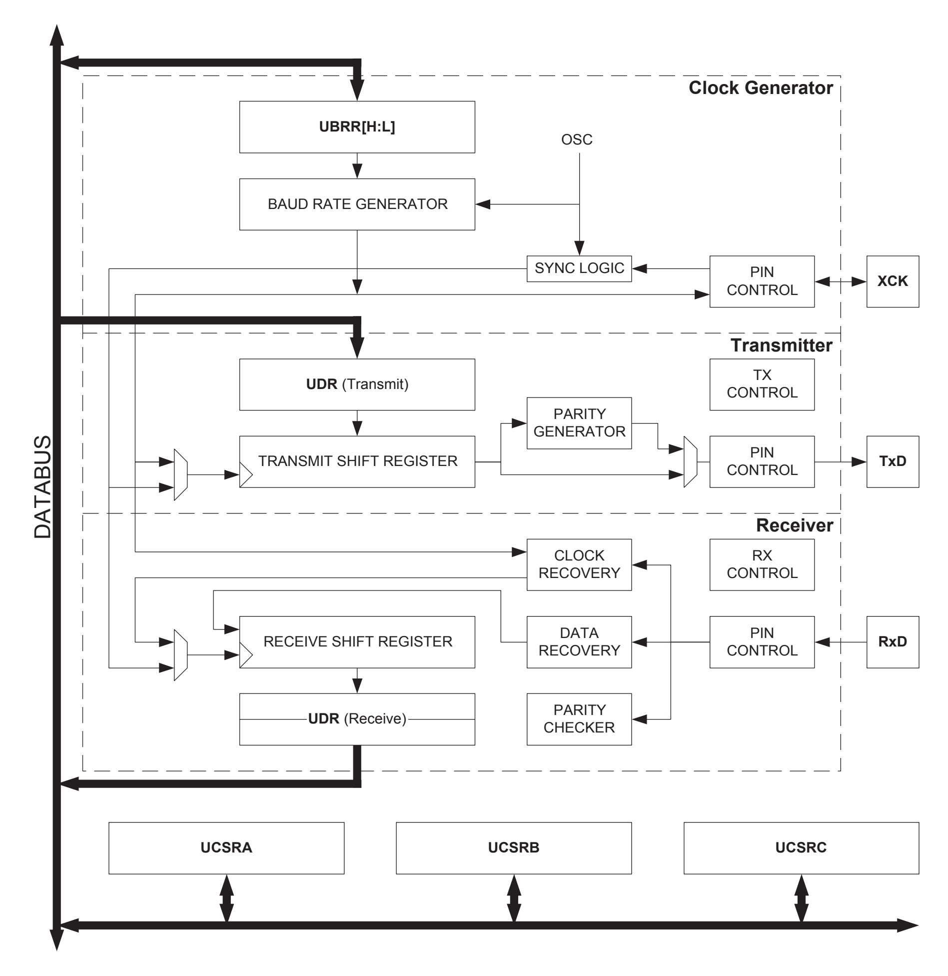
综述

Figure 79 为简化的 USART 转发器。CPU 可以访问的 I/O 寄存器和 I/O 引脚以粗体表示。





Figure 79. USART 方框图



Note: 请参考 P 2Figure 1 , P 73Table 36 和 P 75Table 39 了解 USART 的引脚分布。

虚线框将 USART 分为了三个主要部分 : 时钟发生器 , 发送器和接收器。控制寄存器由三个单元共享。时钟发生器包括同步从机操作用来与外部输入时钟进行同步的逻辑 , 以及波特率发生器。XCK (发送器时钟) 引脚用于同步发送模式。发送器包括单个写缓冲器 , 串行移位寄存器 , 奇偶发生器以及处理不同的帧格式所需的控制逻辑。写缓冲器可以保持连续发送数据而不会在数据帧之间引入延迟。由于接收器具有时钟和数据恢复单元 , 它是 USART 模块中最复杂的。恢复单元用于异步数据的接收。除了恢复单元 , 接收器还包括奇偶校验 , 控制逻辑 , 移位寄存器和两个接收缓冲器 UDR。接收器支持与发送器相同的帧格式 , 而且可以检测帧错误 , 数据过速和奇偶校验错误。

AVR USART 和 AVR UART - 兼容性

USART 在如下方面与 AVR UART 完全兼容：

- 所有 USART 寄存器的位定义
- 波特率发生器
- 发送器操作
- 发送缓冲器的功能
- 接收器操作

然而，接收器缓冲器有两个方面的改进，在某些特殊情况下会影响兼容性：

- 增加了一个缓冲器。两个缓冲器的操作好象是一个循环的 FIFO。因此对于每个接收到的数据只能读一次！更重要的是错误标志 FE 和 DOR，以及第 9 个数据位 RXB8 与数据一起存放于接收缓冲器。因此必须在读取 UDR 寄存器之前访问状态标志位。否则将丢失错误状态。
- 现在接收移位寄存器可以作为第三级缓冲了。其意义是在两个缓冲器都没有空的时候，将数据保存于串行移位寄存器之中 (参见 Figure 79)，直到检测到新的起始位。从而增强了 USART 抵抗数据过速 (DOR) 的能力。

下面的控制位的名称做了改动，但是功能和在寄存器中的位置并没有改变：

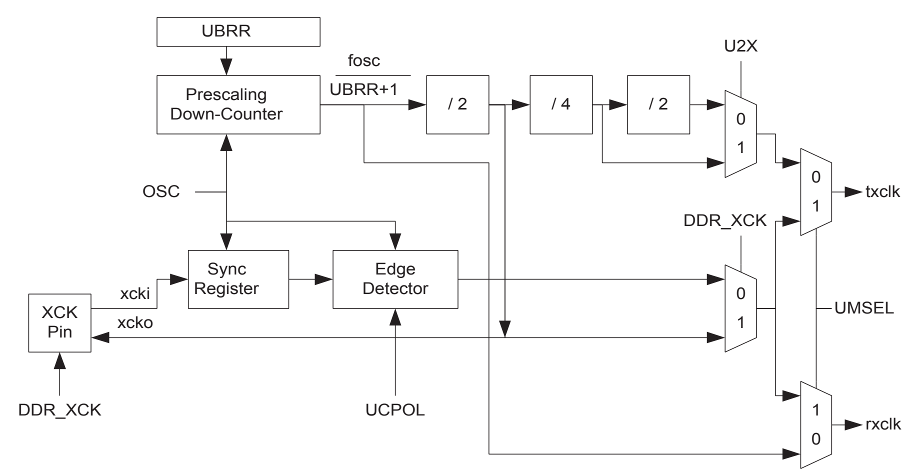
- CHR9 改变为 UCSZ2
- OR 改变为 DOR

时钟产生

时钟产生逻辑为发送器和接收器产生基础时钟。 USART 支持 4 种模式的时钟：正常的异步模式，倍速的异步模式，主机同步模式，以及从机同步模式。 USART 控制和状态寄存器 C (UCSRC) 用于选择异步模式和同步模式。倍速模式 （ 只适用于异步模式 ）受控于 UCSRA 寄存器的 U2X。使用同步模式 (UMSEL = 1) 时， XCK 的数据方向寄存器 (DDR_XCK)决定时钟源是由内部产生(主机模式)还是由外部生产(从机模式)。仅在同步模式下 XCK 有效。

Figure 80 为时钟产生逻辑的框图。

Figure 80. 时钟产生逻辑框图



信号说明：

txclk 发送器时钟 (内部信号)。

rxclk 接收器基础时钟 (内部信号)。

xcki 由 XCK 引脚输入 (内部信号)，用于同步从机操作。

xcko 输出到 XCK 引脚的时钟 (内部信号). 用于同步主机操作。



fosc XTAL 频率 (系统时钟)。

片内时钟产生 - 波特率发生器

内部时钟用于异步模式与同步主机模式，请参见 Figure 80。

USART 的波特率寄存器 UBRR 和降序计数器相连接，一起构成可编程的预分频器或波特率发生器。降序计数器对系统时钟计数，当其计数到零或 UBRRL 寄存器被写时，会自动装入 UBRR 寄存器的值。当计数到零时产生一个时钟，该时钟作为波特率发生器的输出时钟，输出时钟的频率为 f_{osc}/(UBRR+1)。发生器对波特率发生器的输出时钟进行 2、 8 或 16 的分频，具体情况取决于工作模式。波特率发生器的输出被直接用于接收器与数据恢复单元。数据恢复单元使用了一个有2、8或16个状态的状态机，具体状态数由UMSEL、U2X 与 DDR_XCK 位设定的工作模式决定。

Table 74给出了计算波特率(位/秒)以及计算每一种使用内部时钟源工作模式的UBRR值的公式。

Table 74. 波特率计算公式

使用模式	波特率的计算公式 [1]	UBRR 值的计算公式
异步正常模式 (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(\mathbf{UBRR}+1)}$	$\mathbf{UBRR} = \frac{f_{OSC}}{16BAUD} - 1$
异步倍速模式 (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(\mathbf{UBRR}+1)}$	$\mathbf{UBRR} = \frac{f_{OSC}}{8BAUD} - 1$
同步主机模式	$BAUD = \frac{f_{OSC}}{2(\mathbf{UBRR}+1)}$	$\mathbf{UBRR} = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. 波特率定义为每秒的位传输速度 (bps)。

BAUD 波特率 (bps)。

f_{OSC} 系统时钟频率。

UBRR UBRRH 与 UBRRL 的数值 (0-4095)。

Table 82 给出了在某些系统时钟频率下对应的 UBRR 数值。

倍速操作 (U2X)

通过设定 UCSRA 寄存器的 U2X 可以使传输速率加倍。该位只对异步工作模式有效。当工作在同步模式时，设置该位为 "0”。

设置该位把波特率分频器的分频值从 16 降到 8，使异步通信的传输速率加倍。此时接收器只使用一半的采样数对数据进行采样及时钟恢复，因此在该模式下需要更精确的系统时钟与更精确的波特率设置。发送器则没有这个要求。

外部时钟

同步从机操作模式由外部时钟驱动，如 Figure 80 所示。

输入到 XCK 引脚的外部时钟由同步寄存器进行采样，用以提高稳定性。同步寄存器的输出通过一个边沿检测器，然后应用于发送器与接收器。这一过程引入了两个 CPU 时钟周期的延时，因此外部 XCK 的最大时钟频率由以下公式限制：

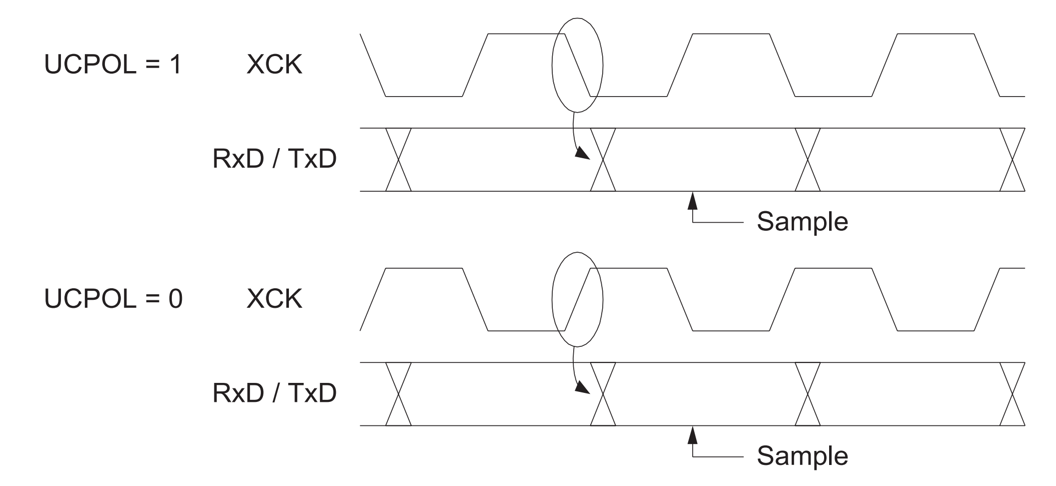
$$f_{XCK} < \frac{f_{OSC}}{4}$$

要注意 f_{osc} 由系统时钟的稳定性决定，为了防止因频率漂移而丢失数据，建议保留足够的裕量。

同步时钟操作

使用同步模式时 (UMSEL = 1)XCK 引脚被用于时钟输入 (从机模式) 或时钟输出 (主机模式)。时钟的边沿、数据的采样与数据的变化之间的关系的基本规律是：在改变数据输出端 TxD 的 XCK 时钟的相反边沿对数据输入端 RxD 进行采样。

Figure 81. 同步模式时的 XCK 时序 .



UCRSC 寄存器的 UCPOL 位确定使用 XCK 时钟的哪个边沿对数据进行采样和改变输出数据。如 Figure 81 所示，当 UCPOL=0 时，在 XCK 的上升沿改变输出数据，在 XCK 的下降沿进行数据采样；当 UCPOL=1 时 ,在 XCK 的下降沿改变输出数据 ,在 XCK 的上升沿进行数据采样。



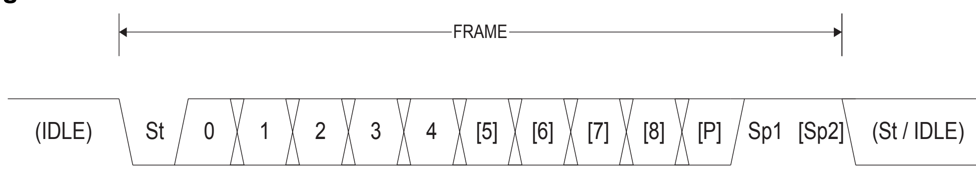
帧格式

串行数据帧由数据字加上同步位 (开始位与停止位) 以及用于纠错的奇偶校验位构成。USART 接受以下 30 种组合的数据帧格式：

- 1 个起始位
- 5、6、7、8 或 9 个数据位
- 无校验位、奇校验或偶校验位
- 1 或 2 个停止位

数据帧以起始位开始；紧接着是数据字的最低位，数据字最多可以有 9 个数据位，以数据的最高位结束。如果使能了校验位，校验位将紧接着数据位，最后是结束位。当一个完整的数据帧传输后，可以立即传输下一个新的数据帧，或使传输线处于空闲状态。 Figure 82 所示为可能的数据帧结构组合。括号中的位是可选的。

Figure 82. 帧格式



St 起始位，总是为低电平。

(n) 数据位 (0 ～ 8)。

P 校验位，可以为奇校验或偶校验。

Sp 停止位，总是为高电平。

IDLE 通讯线上没有数据传输 (RxD 或 TxD)，线路空闲时必须为高电平。

数据帧的结构由 UCSRB 和 UCSRC 寄存器中的 UCSZ2:0、UPM1:0 与 USBS 设定。接收与发送使用相同的设置。设置的任何改变都可能破坏正在进行的数据传送与接收。

USART 的字长位 UCSZ2:0 确定了数据帧的数据位数；校验模式位 UPM1:0 用于使能与决定校验的类型； USBS 位设置帧有一位或两位结束位。接收器忽略第二个停止位，因此帧错误 (FE) 只在第一个结束位为 "0" 时被检测到。

计算奇偶校验位

校验位的计算是对数据的各个位进行异或运算。如果选择了奇校验，则异或结果还需要取反。校验位与数据位的关系如下：

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

P_{even} 偶校验结果

P_{odd} 奇校验位结果

d_n 第 n 个数据位

校验位处于最后一个数据位与第一个停止位之间。

USART 初始化

进行通信之前首先要对 USART 进行初始化。初始化过程通常包括波特率的设定，帧结构的设定，以及根据需要使能接收器或发送器。对于中断驱动的 USART 操作，在初始化时首先要清零全局中断标志位 (全局中断被屏蔽)。

重新改变 USART 的设置应该在没有数据传输的情况下进行。 TXC 标志位可以用来检验一个数据帧的发送是否已经完成， RXC 标志位可以用来检验接收缓冲器中是否还有数据未读出。在每次发送数据之前 (在写发送数据寄存器 UDR 前)TXC 标志位必须清零。

以下是 USART 初始化程序示例。例程采用了轮询 (中断被禁用) 的异步操作，而且帧结构是固定的。波特率作为函数参数给出。在汇编程序里波特率参数保存于寄存器 r17:r16。

汇编代码例程 (1)

```
USART_Init:
  ; 设置波特率
  out   UBRRH, r17
  out   UBRRL, r16
  ; 接收器与发送器使能
  ldi   r16, (1<<RXEN)|(1<<TXEN)
  out   UCSRB,r16
  ;设置帧格式： 8 个数据位， 2 个停止位
  ldi   r16, (1<<USBS)|(3<<UCSZ0)
  out   UCSRC,r16
  ret
```

C 代码例程 (1)

```
void USART_Init( unsigned int baud )
{
  /* 设置波特率*/
  UBRRH = (unsigned char)(baud>>8);
  UBRRL = (unsigned char)baud;
  /* 接收器与发送器使能*/
  UCSRB = (1<<RXEN)|(1<<TXEN);
  /* 设置帧格式： 8 个数据位， 2 个停止位 */
  UCSRC = (1<<USBS)|(3<<UCSZ0);
}
```

Note: 1. 本代码假定已经包含了合适的头文件。
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

更高级的初始化程序可将帧格式作为参数、禁止中断等等。然而许多应用程序使用固定的波特率与控制寄存器。此时初始化代码可以直接放在主程序中，或与其它 I/O 模块的初始化代码组合到一起。

发送数据 - USART 发送器

置位 UCSRB 寄存器的发送允许位 TXEN 将使能 USART 的数据发送。使能后 TxD 引脚的通用 I/O 功能即被 USART 功能所取代，成为发送器的串行输出引脚。发送数据之前要设置好波特率、工作模式与帧结构。如果使用同步发送模式，施加于 XCK 引脚上的时钟信号即为数据发送的时钟。

以 5 到 8 个数据位的方式发送帧

将需要发送的数据加载到发送缓存器将启动数据发送。加载过程即为 CPU 对 UDR 寄存器的写操作。当移位寄存器可以发送新一帧数据时，缓冲的数据将转移到移位寄存器。当移位寄存器处于空闲状态 (没有正在进行的数据传输)，或前一帧数据的最后一个停止位传送结束，它将加载新的数据。一旦移位寄存器加载了新的数据，就会按照设定的波特率完成数据的发送。



以下程序给出一个对 UDRE 标志采用轮询方式发送数据的例子。当发送的数据少于 8 位时，写入 UDR 相应位置的高几位将被忽略。当然，执行本段代码之前首先要初始化 USART。在汇编代码中要发送的数据存放于 R16。

汇编代码例程 (1)

```
USART_Transmit:
  ; 等待发送缓冲器为空
  sbis  UCSRA,UDRE
  rjmp  USART_Transmit
  ; 将数据放入缓冲器，发送数据
  out   UDR,r16
  ret
```

C 代码例程 (1)

```
void USART_Transmit( unsigned char data )
{
  /* 等待发送缓冲器为空 */
  while ( !( UCSRA & (1<<UDRE)) )
        ;
  /* 将数据放入缓冲器，发送数据 */
  UDR = data;
}
```

Note: 1. 本代码假定已经包含了合适的头文件。
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、 “STS”、 “SBRS”、 “SBRC”、 “SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、 “OUT”、 “SBIS”、 “SBIC”、 “CBI” 与 “SBI” 指令。

这个程序只是在载入新的要发送的数据前，通过检测 UDRE 标志等待发送缓冲器为空。如果使用了数据寄存器空中断，则数据写入缓冲器的操作在中断程序中进行。

以 9 个数据位的方式发送帧

如果发送 9 位数据的数据帧 (UCSZ = 7)，应先将数据的第 9 位写入寄存器 UCSRB 的 TXB8，然后再将低8位数据写入发送数据寄存器UDR。以下程序给出发送9位数据的数据帧例子。在汇编代码中要发送的数据存放在 R17:R16 寄存器中。

汇编代码例程 (1)

```
USART_Transmit:
  ; 等待发送缓冲器为空
  sbis  UCSRA,UDRE
  rjmp  USART_Transmit
  ; 将第 9 位从 r17  中复制到 TXB8
  cbi   UCSRB,TXB8
  sbrc  r17,0
  sbi   UCSRB,TXB8
  ; 将低 8 位数据放入缓冲器，发送数据
  out   UDR,r16
  ret
```

C 代码例程

```
void USART_Transmit( unsigned int data )
{
  /* 等待发送缓冲器为空 */
  while ( !( UCSRA & (1<<UDRE))) )
        ;
  /* 将第 9 位复制到 TXB8 */
  UCSRB &= ~(1<<TXB8);
  if ( data & 0x0100 )
    UCSRB |= (1<<TXB8);
  /* 将数据放入缓冲器，发送数据 */
  UDR = data;
}
```

Note: 1. 这些函数均为通用函数。如果 UCSRB 的内容在应用中是固定的，函数可以进一步优化。例如，初始化后只使用 UCSRB 寄存器的 TXB8 位。
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、 “STS”、 “SBRS”、 “SBRC”、 “SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、 “OUT”、 “SBIS”、 “SBIC”、 “CBI” 与 “SBI” 指令。

第 9 位数据在多机通信中用于表示地址帧，在同步通信中可以用于协议处理。

发送器标志和中断

USART 发送器有两个标志位：USART 数据寄存器空标志 UDRE 及传输结束标志 TXC，两个标志位都可以产生中断。

数据寄存器空 UDRE 标志位表示发送缓冲器是否可以接受一个新的数据。该位在发送缓冲器空时被置 "1”；当发送缓冲器包含需要发送的数据时清零。为与将来的器件兼容，写 UCSRA 寄存器时该位要写 "0”。

当 UCSRB 寄存器中的数据寄存器空中断使能位 UDRIE 为 "1” 时，只要 UDRE 被置位 (且全局中断使能)，就将产生 USART 数据寄存器空中断请求。对寄存器 UDR 执行写操作将清零 UDRE。当采用中断方式的传输数据时，在数据寄存器空中断服务程序中必须写一个新的数据到 UDR 以清零 UDRE ；或者是禁止数据寄存器空中断。否则一旦该中断程序结束，一个新的中断将再次产生。

当整个数据帧移出发送移位寄存器，同时发送缓冲器中又没有新的数据时，发送结束标志 TXC 置位。TXC 在传送结束中断执行时自动清零，也可在该位写 "1” 来清零。TXC 标志位



对于采用如 RS-485 标准的半双工通信接口十分有用。在这些应用里，一旦传送完毕，应用程序必须释放通信总线并进入接收状态。

当 UCSRB 上的发送结束中断使能位 TXCIE 与全局中断使能位均被置为 "1" 时，随着 TXC 标志位的置位， USART 发送结束中断将被执行。一旦进入中断服务程序， TXC 标志位即被自动清零，中断处理程序不必执行 TXC 清零操作。

产生奇偶校验位

奇偶校验产生电路为串行数据帧生成相应的校验位。校验位使能 (UPM1 = 1) 时，发送控制逻辑电路会在数据的最后一位与第一个停止位之间插入奇偶校验位。

禁止发送器

TXEN 清零后，只有等到所有的数据发送完成后发送器才能够真正禁止，即发送移位寄存器与发送缓冲寄存器中没有要传送的数据。发送器禁止后，TxD 引脚恢复其通用 I/O 功能。

接收数据 - USART 接收器

置位 UCSRB 寄存器的接收允许位 (RXEN) 即可启动 USART 接收器。接收器使能后 RxD 的普通引脚功能被 USART 功能所取代，成为接收器的串行输入口。进行数据接收之前首先要设置好波特率、操作模式及帧格式。如果使用同步操作， XCK 引脚上的时钟被用为传输时钟。

以 5 到 8 个数据位的方式接收帧

一旦接收器检测到一个有效的起始位，便开始接收数据。起始位后的每一位数据都将以所设定的波特率或 XCK 时钟进行接收，直到收到一帧数据的第一个停止位。接收到的数据被送入接收移位寄存器。第二个停止位会被接收器忽略。接收到第一个停止位后，接收移位寄存器就包含了一个完整的数据帧。这时移位寄存器中的内容将被转移到接收缓冲器中。通过读取 UDR 就可以获得接收缓冲器的内容的。

以下程序给出一个对 RXC 标志采用轮询方式接收数据的例子。当数据帧少于 8 位时，从 UDR 读取的相应的高几位为 0。当然，执行本段代码之前首先要初始化 USART。

汇编代码例程 [1]

```
USART_Receive:
  ; 等待接收数据
  sbis  UCSRA, RXC
  rjmp  USART_Receive
  ; 从缓冲器中获取并返回数据
  in    r16, UDR
  ret
```

C 代码例程 [1]

```
unsigned char USART_Receive( void )
{
  /* 等待接收数据 */
  while ( !(UCSRA & (1<<RXC)) )
        ;
  /* 从缓冲器中获取并返回数据 */
  return UDR;
}
```

Note: 1. 本代码假定已经包含了相应的头文件。
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、 “STS”、 “SBRS”、 “SBRC”、 “SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、 “SBIC”、 “CBI” 与 “SBI” 指令。

在读缓冲器并返回之前，函数通过检查 RXC 标志来等待数据送入接收缓冲器。



以 9 个数据位的方式接收帧

如果设定了 9 位数据的数据帧 (UCSZ=7)，在从 UDR 读取低 8 位**之前**必须首先读取寄存器 UCSRB 的 RXB8 以获得第 9 位数据。这个规则同样适用于状态标志位 FE、 DOR 及 UPE。状态通过读取UCSRA获得，数据通过UDR获得。读取UDR存储单元会改变接收缓冲器 FIFO 的状态，进而改变同样存储在 FIFO 中的 TXB8、FE、 DOR 及 UPE 位。

接下来的代码示例展示了一个简单的USART接收函数，说明如何处理9位数据及状态位。

汇编代码例程 (1)

```
USART_Receive:
  ; 等待接收数据
  sbis  UCSRA, RXC
  rjmp  USART_Receive
  ; 从缓冲器中获得状态、第 9 位及数据
  in    r18, UCSRA
  in    r17, UCSRB
  in    r16, UDR
  ; 如果出错，返回 -1
  andi  r18,(1<<FE)|(1<<DOR)|(1<<UPE)
  breq  USART_ReceiveNoError
  ldi   r17, HIGH(-1)
  ldi   r16, LOW(-1)
USART_ReceiveNoError:
  ; 过滤第 9 位数据，然后返回
  lsr   r17
  andi  r17, 0x01
  ret
```

C 代码例程 (1)

```
unsigned int USART_Receive( void )
{
  unsigned char status, resh, resl;
  /* 等待接收数据 */
  while ( !(UCSRA & (1<<RXC)) )
        ;
  /* 从缓冲器中获得状态、第 9 位及数据 */
  status = UCSRA;
  resh = UCSRB;
  resl = UDR;
  /* 如果出错，返回 -1 */
  if ( status & (1<<FE)|(1<<DOR)|(1<<UPE) )
    return -1;
  /* 过滤第 9 位数据，然后返回 */
  resh = (resh >> 1) & 0x01;
  return ((resh << 8) | resl);
}
```

Note: 1. 本代码假定已经包含了相应的头文件。
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、 “STS”、 “SBRS”、 “SBRC”、 “SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、 “OUT”、 “SBIS”、 “SBIC”、 “CBI” 与 “SBI” 指令。

上述例子在进行任何计算之前将所有的 I/O 寄存器的内容读到寄存器文件中。这种方法优化了对接收缓冲器的利用。它尽可能早地释放了缓冲器以接收新的数据。

接收完成标志和中断

USART 接收器有一个标志用来指明接收器的状态。

接收结束标志 (RXC) 用来说明接收缓冲器中是否有未读出的数据。当接收缓冲器中有未读出的数据时，此位为 1，当接收缓冲器空时为 0(即不包含未读出的数据)。如果接收器被禁止 (RXEN = 0)，接收缓冲器会被刷新，从而使 RXC 清零。

置位 UCSRB 的接收结束中断使能位 (RXCIE) 后，只要 RXC 标志置位 (且全局中断只能) 就会产生 USART 接收结束中断。使用中断方式进行数据接收时，数据接收结束中断服务程序程序必须从 UDR 读取数据以清 RXC 标志，否则只要中断处理程序一结束，一个新的中断就会产生。

接收器错误标志

USART 接收器有三个错误标志: : 帧错误 (FE)、数据溢出 (DOR) 及奇偶校验错 (UPE)。它们都位于寄存器 UCSRA。错误标志与数据帧一起保存在接收缓冲器中。由于读取 UDR 会改变缓冲器， UCSRA 的内容必须在读接收缓冲器 (UDR) 之前读入。错误标志的另一个同一性是它们都不能通过软件写操作来修改。但是为了保证与将来产品的兼容性，对执行写操作是必须对这些错误标志所在的位置写 "0“。所有的错误标志都不能产生中断。

帧错误标志 (FE) 表明了存储在接收缓冲器中的下一个可读帧的第一个停止位的状态。停止位正确 (为 1) 则 FE 标志为 0，否则 FE 标志为 1。这个标志可用来检测同步丢失、传输中断，也可用于协议处理。UCSRC 中 USBS 位的设置不影响 FE 标志位，因为除了第一位，接收器忽略所有其他的停止位。为了与以后的器件相兼容，写 UCSRA 时这一位必须置 0。

数据溢出标志 (DOR) 表明由于接收缓冲器满造成了数据丢失。当接收缓冲器满（包含了两个数据），接收移位寄存器又有数据，若此时检测到一个新的起始位，数据溢出就产生了。DOR 标志位置位即表明在最近一次读取 UDR 和下一次读取 UDR 之间丢失了一个或更多的数据帧。为了与以后的器件相兼容，写 UCSRA 时这一位必须置 0。当数据帧成功地从移位寄存器转入接收缓冲器后， DOR 标志被清零。

奇偶校验错标志 (UPE) 指出，接收缓冲器中的下一帧数据在接收时有奇偶错误。如果不使能奇偶校验，那么 UPE 位应清零。为了与以后的器件相兼容，写 UCSRA 时这一位必须置 0。细节请参照 P 160“ 计算奇偶校验位 ” 及 P 168“ 奇偶校验器 ” 。



奇偶校验器

奇偶校验模式位UPM1置位将启动奇偶校验器。校验的模式(偶校验还是奇校验)由UPM0确定。奇偶校验使能后，校验器将计算输入数据的奇偶并把结果与数据帧的奇偶位进行比较。校验结果将与数据和停止位一起存储在接收缓冲器中。这样就可以通过读取奇偶校验错误标志位 (UPE) 来检查接收的帧中是否有奇偶错误。

如果下一个从接收缓冲器中读出的数据有奇偶错误，并且奇偶校验使能 (UPM1 = 1)，则 UPE 置位。直到接收缓冲器 (UDR) 被读取，这一位一直有效。

接收器禁用

与发送器对比，禁止接收器即刻起作用。正在接收的数据将丢失。禁止接收器 (RXEN 清零) 后，接收器将不再占用 RxD 引脚；接收缓冲器 FIFO 也会被刷新。缓冲器中的数据将丢失。

刷新接收缓冲器

禁止接收器时缓冲器 FIFO 被刷新，缓冲器被清空。导致未读出的数据丢失。如果由于出错而必须在正常操作下刷新缓冲器，则需要一直读取 UDR 直到 RXC 标志清零。下面的代码展示了如何刷新接收缓冲器。

汇编代码例程 [(1)]

```
USART_Flush:
  sbis  UCSRA, RXC
  ret
  in    r16, UDR
  rjmp  USART_Flush
```

C 代码例程 [(1)]

```
void USART_Flush( void )
{
  unsigned char dummy;
  while ( UCSRA & (1<<RXC) ) dummy = UDR;
}
```

Note: 1. 本代码假定已经包含了相应的头文件。
当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、“OUT”、“SBIS”、“SBIC”、“CBI” 与 “SBI” 指令。

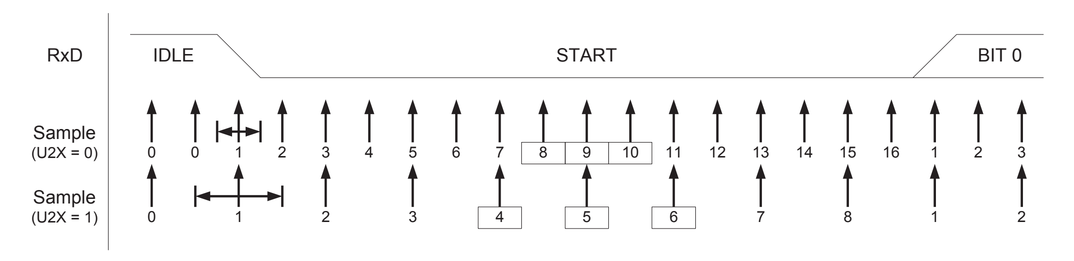
接收异步数据

USART 有一个时钟恢复单元和数据恢复单元用来处理异步数据接收。时钟恢复逻辑用于同步从 RxD 引脚输入的异步串行数据和内部的波特率时钟。数据恢复逻辑采集数据，并通过一低通滤波器过滤所输入的每一位数据，从而提高接收器的抗干扰性能。异步接收的工作范围依赖于内部波特率时钟的精度、帧输入的速率及一帧所包含的位数。

恢复异步时钟

时钟恢复逻辑将输入的串行数据帧与内部时钟同步起来。 Figure 83 展示了对输入数据帧起始位的采样过程。普通工作模式下采样率是波特率的 16 倍，倍速工作模式下则为波特率的 8 倍。水平箭头表示由于采样而造成的同步的变化。使用倍速模式 (U2X = 1) 时同步变化时间更长。 RxD 线空闲 (即没有任何通讯活动) 时，采样值为 0。

Figure 83. 起始位采样

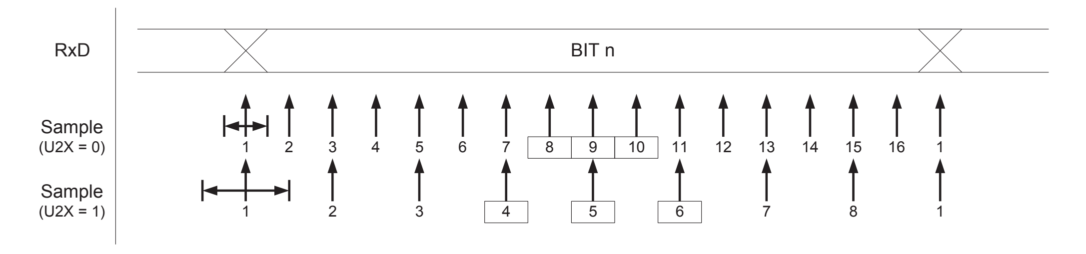


当时钟恢复电路检测到 RxD 线上一个由高 (空闲) 到低 (开始) 的电平跳变时，起始位检测序列即被启动。如图所示，我们用采样 1 表示第一个 0 采样。然后，时钟恢复逻辑用采样 8、 9、 10(普通模式)，或采样 4、 5、 6(倍速模式)，来判断是否接收到一个正确的起始位。如果这三个采样中的两个或更多个是逻辑高电平（多数表决），起始位会被视为毛刺噪声而被拒绝接受，接收器等待下一个由高到低的电平转换。如果检测到一个有效的起始位，时钟恢复逻辑即被同步并开始接收数据。每一个起始位都会引发同样的同步过程。

恢复异步数据

接收时钟与起始位同步之后，数据恢复工作可开始了。数据恢复单元使用一个状态机来接收每一个数据位。这个状态机在普通模式下具有 16 个状态，在倍速模式下具有 8 个状态。 Figure 84 演示了对数据位和奇偶位的采样。每个采样点都被赋予了一个数字，这个数字等于数据恢复单元当前的状态序号。

Figure 84. 数据及奇偶位的采样

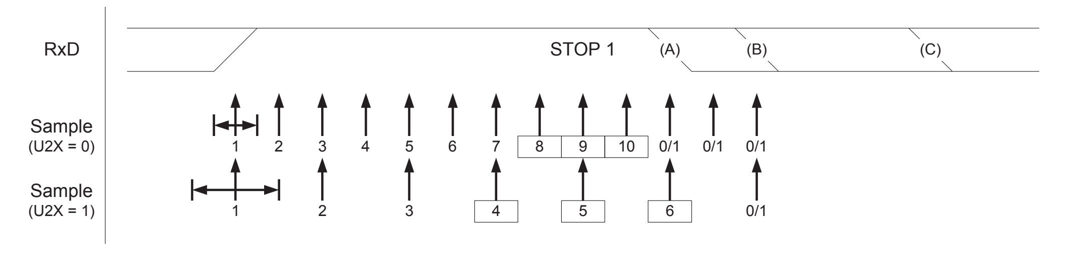


确定接收到的数据位的逻辑电平的方法为多数表决法。表决对象即为三个在数据位中心获得的采样。为了强调这些采样，图中采样序号被包含小方框中。多数表决是这样工作的：如果有 2 个或所有 3 个采样值都是高电平，那么接收位就为逻辑 1。如果 2 个或所有 3个采样值都是低电平，那么接收位就被为逻辑0。对从RxD引脚输入的信号来说，多数表决的作用就象是一个低通滤波。数据恢复过程重复进行，直到接收到一个完整的数据帧。其中也包含了第一个停止位。接收器将忽略其他的停止位。 Figure 85 说明了停止位的采样，以及下一帧信号起始位最早可能出现的情况。





Figure 85. 停止位及下一个起始位采样



多数表决对停止位同样有效。若停止位为逻辑 0，那么帧错误标志 FE 置位。

如果电平再一次出现了从高到低的跳变，说明紧接着上一个数据帧来了新的数据帧。在普通模式中，第一个低电平的采样点可以发生在 Figure 85 的 A 点。在倍速工作模式下第一个低电平采样点必须延迟到 B 点， C 点则为完整停止位的结束位置。对起始位的及早检测将影响接收器的工作范围。

异步工作范围

接收器的工作范围取决于接收到的数据速率及内部波特率之间的不匹配程度。如果发送器以过快或过慢的比特率传输数据帧，或者接收器内部产生的波特率没有相同的频率 (见 Table 75)，那么接收器就无法与起始位同步。

下面的公式可用来计算数据输入速率与内部接收器波特率的比值。

$$R_{slow} = \frac{(D+1)S}{S-1+D\cdot S+S_F} \qquad R_{fast} = \frac{(D+2)S}{(D+1)S+S_M}$$

D 字符长度及奇偶位长度的总和 (D = 5 到 10 位)。

S 每一位的采样数。普通模式下 S = 16，倍速模式下 S = 8 。

S_F 用于多数表决的第一个采样序号。普通模式下 $S_F = 8$ ，倍速模式下 $S_F = 4$。

S_M 用于多数表决的中间采样序号。普通模式下 $S_M = 9$ ，倍速模式下 $S_M = 5$ 。

R_{slow} 是可接受的、最慢的数据输入速率与接收器波特率的比值； R_{fast} 是可接受的、最快的数据输入速率与接收器波特率的比值。

Table 75 和 Table 76 列出了容许的最大接收器波特率误差。需要注意的是，普通模式下波特率允许有更大的变化范围。

Table 75. 普通模式下推荐的最大接收器波特率误差范围 (U2X = 0)

D # (数据 + 奇偶位)	R_{slow} %	R_{fast} %	最大的总误差 (%)	推荐的最大接收器误差 (%)
5	93,20	106,67	+6.67/-6.8	± 3.0
6	94,12	105,79	+5.79/-5.88	± 2.5
7	94,81	105,11	+5.11/-5.19	± 2.0
8	95,36	104,58	+4.58/-4.54	± 2.0
9	95,81	104,14	+4.14/-4.19	± 1.5
10	96,17	103,78 %	+3.78/-3.83	± 1.5

Table 76. 倍速率模式下推荐的最大接收器波特率误差范围 (U2X = 1)

D # 数据 + 奇偶位	R_{slow} (%)	R_{fast} (%)	最大的总误差 (%)	推荐的最大接收器误差 (%)
5	94,12	105,66	+5.66/-5.88	± 2.5
6	94,92	104,92	+4.92/-5.08	± 2.0
7	95,52	104,35	+4.35/-4.48	± 1.5
8	96,00	103,90	+3.90/-4.00	± 1.5
9	96,39	103,53	+3.53/-3.61	± 1.5
10	96,70	103,23	+3.23/-3.30	± 1.0

上述推荐的最大接收波特率误差是在假定接收器和发送器对最大总误差具有同等贡献的前提下得出的。

产生接收器波特率误差的可能原因有两个。首先，接收器系统时钟 (XTAL) 的稳定性于电压范围及工作温度有关。使用晶振来产生系统时钟时一般不会有此问题，但对于谐振器而言，根据谐振器不同的误差容限，系统时钟可能有超过 2% 的偏差。第二个误差的原因就好控制多了。波特率发生器不一定能够通过对系统时钟的分频得到恰好的波特率。此时可以调整 UBRR 值，使得误差低至可以接受。

多处理器通讯模式

置位 UCSRA 的多处理器通信模式位 (MPCM) 可以对 USART 接收器接收到的数据帧进行过滤。那些没有地址信息的帧将被忽略，也不会存入接收缓冲器。在一个多处理器系统中，处理器通过同样的串行总线进行通信，这种过滤有效的减少了需要 CPU 处理的数据帧的数量。MPCM 位的设置不影响发送器的工作 ,但在使用多处理器通信模式的系统中，它的使用方法会有所不同。

如果接收器所接收的数据帧长度为 5 到 8 位 ,那么第一个停止位表示这一帧包含的是数据还是地址信息。如果接收器所接收的数据帧长度为 9 位 ,那么由第 9 位 (RXB8) 来确定是数据还是地址信息。如果确定帧类型的位 (第一个停止位或第 9 个数据位) 为 1，那么这是地址帧，否则为数据帧。

在多处理器通信模式下 ,多个从处理器可以从一个主处理器接收数据。首先要通过解码地址帧来确定所寻址的是哪一个处理器。如果寻址到某一个处理器，它将正常接收后续的数据，而其他的从处理器会忽略这些帧直到接收到另一个地址帧。

使用 MPCM

对于一个作为主机的处理器来说，它可以使用 9 位数据帧格式 (UCSZ = 7)。如果传输的是一个地址帧 (TXB8 = 1) 就将第 9 位 (TXB8) 置 1，如果是一个数据帧 (TXB = 0) 就将它清零。在这种帧格式下，从处理器必须工作于 9 位数据帧格式。

下面即为在多处理器通信模式下进行数据交换的步骤：

1. 所有从处理器都工作在多处理器通信模式 (UCSRA 寄存器的 MPCM 置位)。
2. 主处理器发送地址帧后，所有从处理器都会接收并读取此帧。从处理器 UCSRA 寄存器的 RXC 正常置位。
3. 每一个从处理器都会读取 UDR 寄存器的内容已确定自己是否被选中。如果选中，就清零 UCSRA 的 MPCM 位，否则它将等待下一个地址字节的到来，并保持 MPCM 为 1。
4. 被寻址的从处理器将接收所有的数据帧，直到收到一个新的地址帧。而那些保持 MPCM 位为 1 的从处理器将忽略这些数据。
5. 被寻址的处理器接收到最后一个数据帧后，它将置位 MPCM，并等待主处理器发送下一个地址帧。然后第 2 步之后的步骤重复进行。

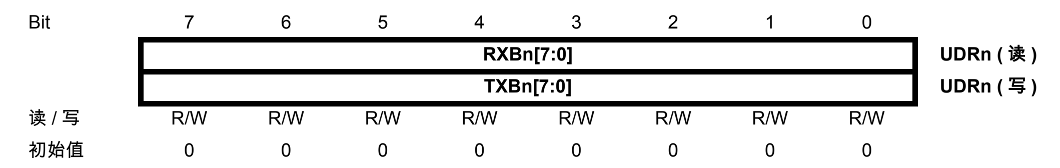


使用 5 至 8 比特的帧格式是可以的，但是不实际，因为接收器必须在使用 n 和 n+1 帧格式之间进行切换。由于接收器和发送器使用相同的字符长度设置，这种设置使得全双工操作变得很困难。如果使用 5 至 8 比特的帧格式，发送器应该设置两个停止位 (USBS = 1)，其中的第一个停止位被用于判断帧类型。

不要使用读 - 修改 - 写指令 (SBI 和 CBI) 来操作 MPCM 位。 MPCM 和 TXC 标志使用相同的 I/O 单元，使用 SBI 或 CBI 指令可能会不小心将它清零。

USART 寄存器说明

USARTn I/O 数据寄存器 - UDRn



Bit	7	6	5	4	3	2	1	0	
	RXBn[7:0]								UDRn (读)
	TXBn[7:0]								UDRn (写)
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

USART 发送数据缓冲寄存器和 USART 接收数据缓冲寄存器共享相同的 I/O 地址，称为 USART 数据寄存器或 UDR。将数据写入 UDR 时实际操作的是发送数据缓冲器存器 (TXB)，读 UDR 时实际返回的是接收数据缓冲寄存器 (RXB) 的内容。

在 5、6、7 比特字长模式下，未使用的高位被发送器忽略，而接收器则将它们设置为 0。

只有当 UCSRA 寄存器的 UDRE 标志置位后才可以对发送缓冲器进行写操作。如果 UDRE 没有置位，那么写入 UDR 的数据会被 USART 发送器忽略。当数据写入发送缓冲器后，若移位寄存器为空，发送器将把数据加载到发送移位寄存器。然后数据串行地从 TxD 引脚输出。

接收缓冲器包括一个两级 FIFO，一旦接收缓冲器被寻址 FIFO 就会改变它的状态。因此不要对这一存储单元使用读 - 修改 - 写指令 (SBI 和 CBI)。使用位查询指令 (SBIC 和 SBIS) 时也要小心，因为这也有可能改变 FIFO 的状态。

USART 控制和状态寄存器 A - UCSRnA

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
读 / 写	R	R/W	R	R	R	R	R/W	R/W	
初始值	0	0	1	0	0	0	0	0	

- **Bit 7 – RXCn: USART 接收结束**

接收缓冲器中有未读出的数据时 RXCn 置位，否则清零。接收器禁止时，接收缓冲器被刷新，导致 RXCn 清零。 RXCn 标志可用来产生接收结束中断 (见对 RXCIEn 位的描述)。

- **Bit 6 – TXCn: USART 发送结束**

发送移位缓冲器中的数据被送出，且当发送缓冲器 (UDRn) 为空时 TXCn 置位。执行发送结束中断时 TXCn 标志自动清零，也可以通过写 1 进行清除操作。TXCn 标志可用来产生发送结束中断 (见对 TXCIEn 位的描述)。

- **Bit 5 – UDREn: USART 数据寄存器空**

UDREn 标志指出发送缓冲器 (UDRn) 是否准备好接收新数据。UDREn 为 1 说明缓冲器为空，已准备好进行数据接收。 UDREn 标志可用来产生数据寄存器空中断 (见对 UDRIEn 位的描述)。

复位后 UDREn 置位，表明发送器已经就绪。

- **Bit 4 – FEn: 帧错误**

如果接收缓冲器接收到的下一个字符有帧错误，即接收缓冲器中的下一个字符的第一个停止位为 0，那么 FEn 置位。这一位一直有效直到接收缓冲器 (UDRn) 被读取。当接收到的停止位为 1 时， FEn 标志为 0。对 UCSRnA 进行写入时，这一位要写 0。

- **Bit 3 – DORn: 数据过速**

数据过速时 DORn 置位。当接收缓冲器满 (包含了两个数据)，接收移位寄存器又有数据，若此时检测到一个新的起始位，数据溢出就产生了。这一位一直有效直到接收缓冲器 (UDRn) 被读取。对 UCSRnA 进行写入时，这一位要写 0。

- **Bit 2 – UPEn: 奇偶校验错误**

当奇偶校验使能 (UPMn1 = 1) , 且接收缓冲器中所接收到的下一个字符有奇偶校验错误时 UPEn 置位。这一位一直有效直到接收缓冲器 (UDRn) 被读取。对 UCSRnA 进行写入时，这一位要写 0。

- **Bit 1 – U2Xn: 倍速发送**

这一位仅对异步操作有影响。使用同步操作时将此位清零。

此位置 1 可将波特率分频因子从 16 降到 8 , 从而有效的将异步通信模式的传输速率加倍。

- **Bit 0 – MPCMn: 多处理器通信模式**

设置此位将启动多处理器通信模式。 MPCMn 置位后， USARTn 接收器接收到的那些不包含地址信息的输入帧都将被忽略。发送器不受 MPCMn 设置的影响。详细信息请参考 P 171“ 多处理器通讯模式 ” 。

USARTn 控制和状态寄存器 B - UCSRnB

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – RXCIEn: 接收结束中断使能**

置位后使能 RXCn 中断。当 RXCIEn 为 1 , 全局中断标志位 SREG 置位，UCSRnA 寄存器的 RXCn 亦为 1 时可以产生 USARTn 接收结束中断。

- **Bit 6 – TXCIE: 发送结束中断使能**

置位后使能 TXCn 中断。当 TXCIEn 为 1 , 全局中断标志位 SREG 置位， UCSRnA 寄存器的 TXCn 亦为 1 时可以产生 USARTn 发送结束中断。

- **Bit 5 – UDRIEn: USART 数据寄存器空中断使能**

置位后使能 UDREn 中断。当 UDRIEn 为 1 , 全局中断标志位 SREG 置位， UCSRnA 寄存器的 UDREn 亦为 1 时可以产生 USARTn 数据寄存器空中断。

- **Bit 4 – RXENn: 接收使能**

置位后将启动 USARTn 接收器。 RxDn 引脚的通用端口功能被 USARTn 功能所取代。禁止接收器将刷新接收缓冲器 , 并使 FEn、 DORn 及 UPEn 标志无效。

- **Bit 3 – TXENn: 发送使能**

置位后将启动将启动 USARTn 发送器。 TxDn 引脚的通用端口功能被 USARTn 功能所取代。 TXENn 清零后 , 只有等到所有的数据发送完成后发送器才能够真正禁止 , 即发送移位寄存器与发送缓冲寄存器中没有要传送的数据。发送器禁止后， TxDn 引脚恢复其通用 I/O 功能。

- **Bit 2 – UCSZn2: 字符长度**

UCSZn2 与 UCSRnC 寄存器的 UCSZn1:0 结合在一起可以设置数据帧所包含的数据位数 (字符长度)。

- **Bit 1 – RXB8n: 接收数据位 8**

对 9 位串行帧进行操作时，RXB8n 是第 9 个数据位。读取 UDRn 包含的低位数据之前首先要读取 RXB8n。

- **Bit 0 – TXB8n: 发送数据位 8**

对 9 位串行帧进行操作时 , TXB8n 是第 9 个数据位。写 UDRn 之前首先要对它进行写操作。

USART 控制和状态寄存器 C - UCSRnC

Bit	7	6	5	4	3	2	1	0	
	–	UMSELn	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC

读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
初始值	0	0	0	0	0	1	1	0

注意该寄存器在 ATmega103 兼容模式下无效。

- **Bit 7 – 保留位**

该位保留。为与未来器件兼容，对 UCSRnC 写入时该位必须写 0 。

- **Bit 6 – UMSELn: USART 模式选择**

通过这一位来选择同步或异步工作模式。

Table 77. UMSELn 设置

UMSELn	模式
0	异步操作
1	同步操作

- **Bit 5:4 – UPMn1:0: 奇偶校验模式**

这两位设置奇偶校验的模式并使能奇偶校验。如果使能了奇偶校验，那么在发送数据，发送器都会自动产生并发送奇偶校验位。对每一个接收到的数据，接收器都会产生一奇偶值，并与UPMn0所设置的值进行比较。如果不匹配，那么就将UCSRnA中的UPEn置位。

Table 78. UPMn 设置

UPMn1	UPMn0	奇偶模式
0	0	禁止
0	1	保留
1	0	偶校验
1	1	奇校验

- **Bit 3 – USBSn: 停止位选择**

通过这一位可以设置停止位的位数。接收器忽略这一位的设置。

Table 79. USBSn 设置

USBSn	停止位位数
0	1-bit
1	2-bits

- **Bit 2:1 – UCSZn1:0: 字符长度**

UCSZn1:0与UCSRnB寄存器的 UCSZn2结合在一起可以设置数据帧包含的数据位数(字符长度)。

Table 80. UCSZn 设置

UCSZn2	UCSZn1	UCSZn0	字符长度
0	0	0	5 位
0	0	1	6 位
0	1	0	7 位
0	1	1	8 位



Table 80. UCSZn 设置

UCSZn2	UCSZn1	UCSZn0	字符长度
1	0	0	保留
1	0	1	保留
1	1	0	保留
1	1	1	9 位

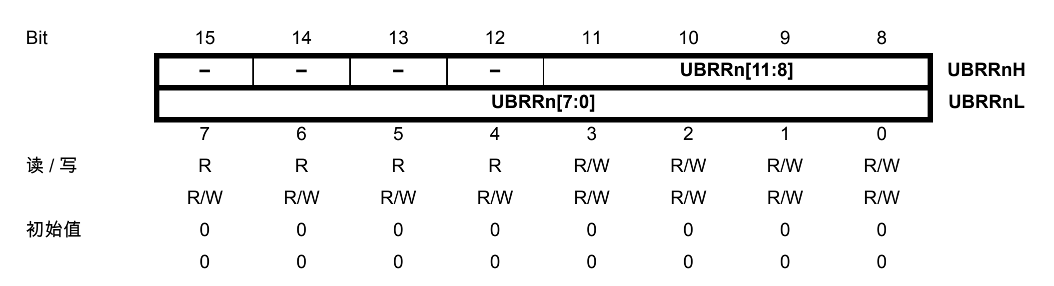
- **Bit 0 – UCPOLn: 时钟极性**

这一位仅用于同步工作模式。使用异步模式时，将这一位清零。 UCPOLn 设置了输出数据的改变和输入数据采样，以及同步时钟 XCKn 之间的关系。

Table 81. UCPOLn 设置

UCPOLn	发送数据的改变 (TxDn 引脚的输出)	接收数据的采样 (RxDn 引脚的输入)
0	XCKn 上升沿	XCKn 下降沿
1	XCKn 下降沿	XCKn 上升沿

USART 波特率寄存器 - UBRRnL 和 UBRRnH



UBRRnH 在 mega103 兼容模式下无效。

- **Bit 15:12 – 保留**

这些位是为以后的使用而保留的。为了与以后的器件兼容，写 UBRRnH 时将这些位清零。

- **Bit 11:0 – UBRRn11:0: USARTn 波特率寄存器**

这个 12 位的寄存器包含了 USARTn 的波特率信息。其中 UBRRnH 包含了 USARTn 波特率高 4 位，UBRRnL 包含了低 8 位。波特率的改变将造成正在进行的数据传输受到破坏。写 UBRRnL 将立即更新波特率分频器。

设置波特率的例子

对标准晶振及谐振器频率来说，异步模式下最常用的波特率可通过 Table 82 中 UBRR 的设置来产生。表中的粗体数据表示由此产生的波特率与目标波特率的偏差不超过 0.5%。更高的误差也是可以接受的，但发送器的抗噪性会降低，特别是需要传输大量数据时 (参看 P 170“ 异步工作范围 ”)。误差可以通过如下公式计算：

$$\text{Error}[\%] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1\right) \bullet 100\%$$

Table 82. 通用振荡器频率下设置 UBRR 的例子

波特率 (bps)	f_{osc} = 1.0000 MHz				f_{osc} = 1.8432 MHz				f_{osc} = 2.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
最大 [(1)]	62.5 kbps		125 kbps		115.2 kbps		230.4 kbps		125 kbps		250 kbps	

1. UBRR = 0, 误差 = 0.0%





Table 83. 通用振荡器频率下设置 UBRR 的例子

波特率 (bps)	f_{osc} = 3.6864 MHz				f_{osc} = 4.0000 MHz				f_{osc} = 7.3728 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4k	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%
19.2k	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%
28.8k	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4k	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6k	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8k	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2k	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4k	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250k	0	-7.8%	1	-7.8%	0	0.0%	1	0.0%	1	-7.8%	3	-7.8%
0.5M	–	–	0	-7.8%	–	–	0	0.0%	0	-7.8%	1	-7.8%
1M	–	–	–	–	–	–	–	–	–	–	0	-7.8%
最大 (1)	230.4 kbps		460.8 kbps		250 kbps		0.5 Mbps		460.8 kbps		921.6 kbps	

1. UBRR = 0, 误差 = 0.0%

Table 84. 通用振荡器频率下设置 UBRR 的例子

波特率 (bps)	f_{osc} = 8.0000 MHz				f_{osc} = 11.0592 MHz				f_{osc} = 14.7456 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	–	–	2	-7.8%	1	-7.8%	3	-7.8%
1M	–	–	0	0.0%	–	–	–	–	0	-7.8%	1	-7.8%
最大 (1)	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

1. UBRR = 0, 误差 = 0.0%

Table 85. 通用振荡器频率下设置 UBRR 的例子

波特率 (bps)	f_{osc} = 16.0000 MHz				f_{osc} = 18.4320 MHz				f_{osc} = 20.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差	UBRR	误差
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	–	–	4	-7.8%	–	–	4	0.0%
1M	0	0.0%	1	0.0%	–	–	–	–	–	–	–	–
最大 (1)	1 Mbps		2 Mbps		1.152 Mbps		2.304 Mbps		1.25 Mbps		2.5 Mbps	

1. UBRR = 0, 误差 = 0.0%

两线串行接口 TWI

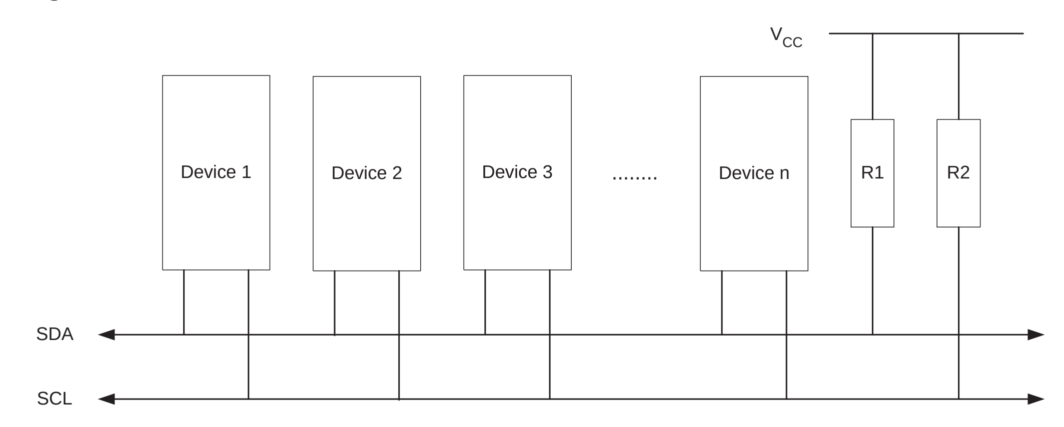
特点

- 简单，但是强大而灵活的通讯接口，只需要两根线
- 支持主机和从机操作
- 器件可以工作于发送器模式或接收器模式
- 7 位地址空间允许有 128 个从机
- 支持多主机仲裁
- 高达 400 kHz 的数据传输率
- 斜率受控的输出驱动器
- 可以抑制总线尖峰的噪声抑制器
- 完全可编程的从机地址以及公共地址
- 睡眠时地址匹配可以唤醒 AVR

两线串行接口总线定义

两线接口 TWI 很适合于典型的处理器应用。 TWI 协议允许系统设计者只用两根双向传输线就可以将 128 个不同的设备互连到一起。这两根线一是时钟 SCL，一是数据 SDA。外部硬件只需要两个上拉电阻，每根线上一个。所有连接到总线上的设备都有自己的地址。TWI 协议解决了总线仲裁的问题。

Figure 86. TWI 总线的连接



TWI 词汇

以下定义将在本节频繁出现。

Table 86. TWI 词汇

单词	说明
主机	启动和停止传输的设备。主机同时要产生 SCL 时钟
从机	被主机寻址的设备
发送器	将数据放到总线上的设备
接收器	从总线读取数据的设备

电气连接

从 Figure 86 可以看出，两根线都通过上拉电阻与正电源连接。所有 TWI 兼容的器件的总线驱动都是漏极开路或集电极开路的。这样就实现了对接口操作非常关键的线与功能。TWI 器件输出为 "0" 时，TWI 总线会产生低电平。当所有的 TWI 器件输出为三态时，总线会输出高电平，允许上拉电阻将电压拉高。注意，为保证所有的总线操作，凡是与 TWI 总线连接的 AVR 器件必须上电。

与总线连接的器件数目受如下条件限制：总线电容要低于 400 pF，而且可以用 7 位从机地址进行寻址。 TWI 详细的电气特性说明请见 P 302“ 两线串行接口特性 ” 。这儿给出了两个不同的规范，一种是总线速度低于 100 kHz，而另外一种是总线速度高达 400 kHz。

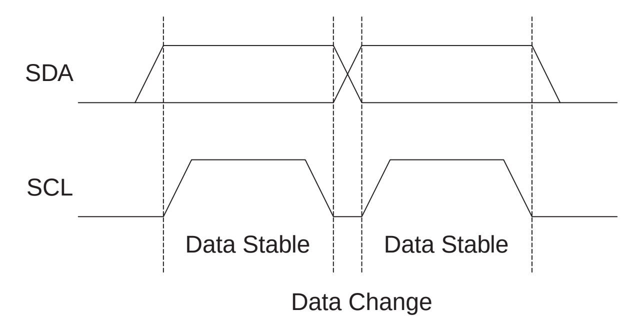


数据传输和帧格式

传输数据 (位)

TWI 总线上数据位的传送与时钟脉冲同步。时钟线为高时，数据线电压必须保持稳定，除非在启动与停止的状态下。

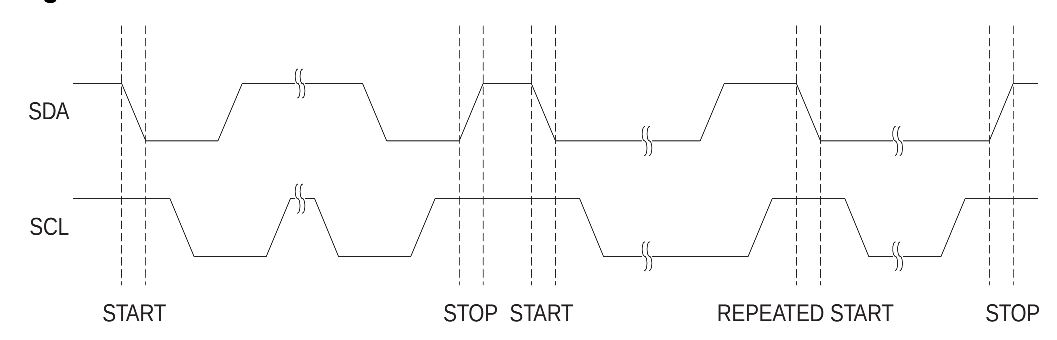
Figure 87. 数据有效性



START/STOP 状态

主机启动与停止数据传输。主机在总线上发出 START 信号以启动数据传输；在总线上发出 STOP 信号以停止数据传输。在 START 与 STOP 状态之间，需要假定总线忙，不允许其它主机控制总线。特例是在 START 与 STOP 状态之间发出一个新的 START 状态。这被称为 REPEATED START 状态，适用于主机在不放弃总线控制的情况下启动新的传送。在 REPEATED START 之后，直到下一个 STOP，需要假定总线处于忙的状态。这与 START 是完全一样的，因此在本手册中，如果没有特殊说明，START 与 REPEATED START均用 START表述。如下所示，START与STOP状态是在SCL线为高时，通过改变SDA 电平来实现的。

Figure 88. START、 REPEATED START 与 STOP 状态



地址数据包格式

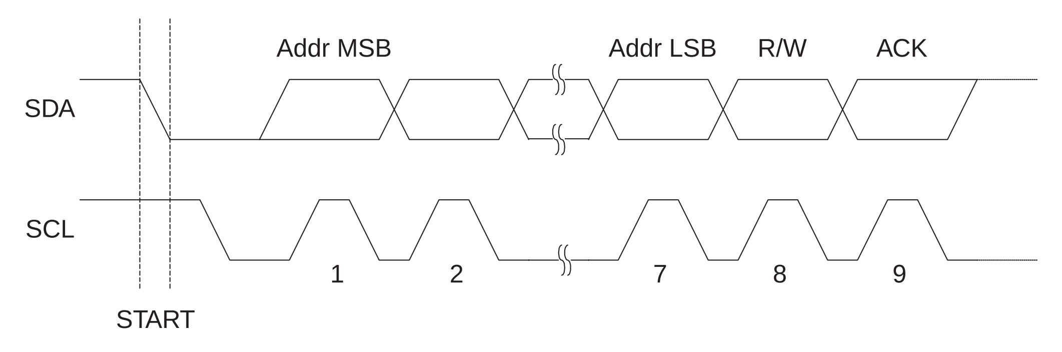
所有在 TWI 总线上传送的地址包均为 9 位，包括 7 位地址位、 1 位 READ/WRITE 控制位与 1 位应答位。如果 READ/WRITE 为 1，则执行读操作；否则执行写操作。从机被寻址后，必须在第九个 SCL (ACK) 周期通过拉低 SDA 作出应答。若该从机忙或有其它原因无法响应主机，则应该在 ACK 周期保持 SDA 为高。然后主机可以发出 STOP 状态或 REPEATED START 状态重新开始发送。地址包包括从机地址与分别称为 SLA+R 或 SLA+W 的 READ 或 WRITE 位。

地址字节的 MSB 首先被发送。从机地址由设计者自由分配，但需要保留地址 0000 000 作为广播地址。

当发送广播呼叫时，所有的从机应在 ACK 周期通过拉低 SDA 作出应答。当主机需要发送相同的信息给多个从机时可以使用广播功能。当 Write 位在广播呼叫之后发送，所有的从机通过在 ACK 周期通过拉低 SDA 作出响应。所有的从机接收到紧跟的数据包。注意在整体访问中发送 Read 位没有意义，因为如果几个从机发送不同的数据会带来总线冲突。

所有形如 1111 xxx 格式的地址都需要保留，以便将来使用。

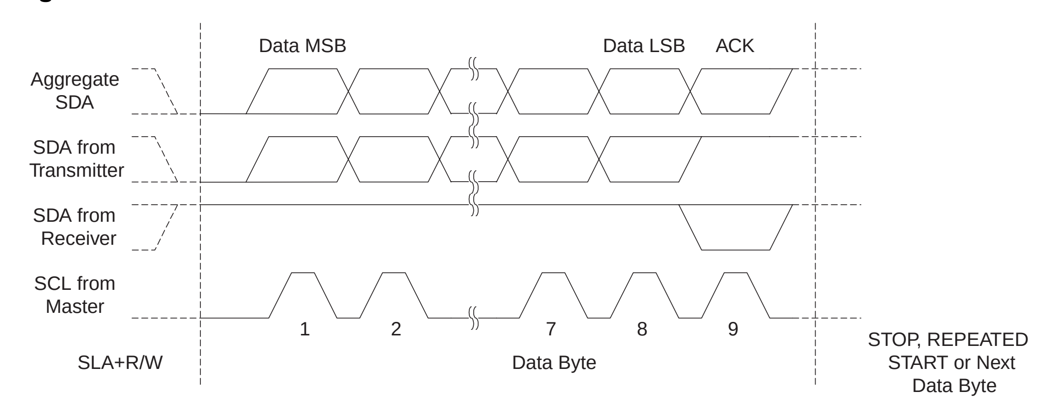
Figure 89. 地址包格式



数据包格式

所有在 TWI 总线上传送的数据包为 9 位长，包括 8 位数据位及 1 位应答位。在数据传送中，主机产生时钟及 START 与 STOP 状态，而接收器响应接收。应答是由从机在第 9 个 SCL 周期拉低 SDA 实现的。如果接收器使 SDA 为高，则发出 NACK 信号。接收器完成接收，或者由于某些原因无法接收更多的数据，应该在收到最后的字节后发出 NACK 来告知发送器。数据的 MSB 首先发送。

Figure 90. 数据包格式

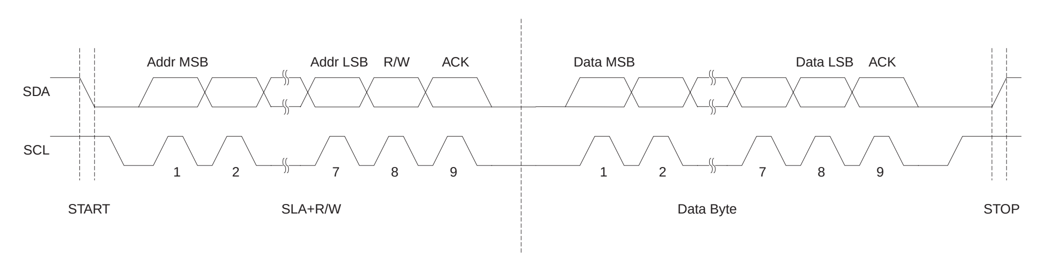


将地址包和数据包组合为一个完整的传输过程

发送主要由 START 状态、SLA+R/W、至少一个数据包及 STOP 状态组成。只有 START 与 STOP 状态的空信息是非法的。可以利用 SCL 的线与功能来实现主机与从机的握手。从机可通过拉低 SCL 来延长 SCL 低电平的时间。当主机设定的时钟速度相对于从机太快，或从机需要额外的时间来处理数据时，这一特性是非常有用的。从机延长 SCL 低电平的时间不会影响 SCL 高电平的时间，因为 SCL 高电平时间是由主机决定的。由上述可知，通过改变 SCL 的占空比可降低 TWI 数据传送速度。

Figure 91 说明了典型的数据传送。注意 SLA+R/W 与 STOP 之间传送的字节数由应用程序的协议决定。

Figure 91. 典型的数据传送



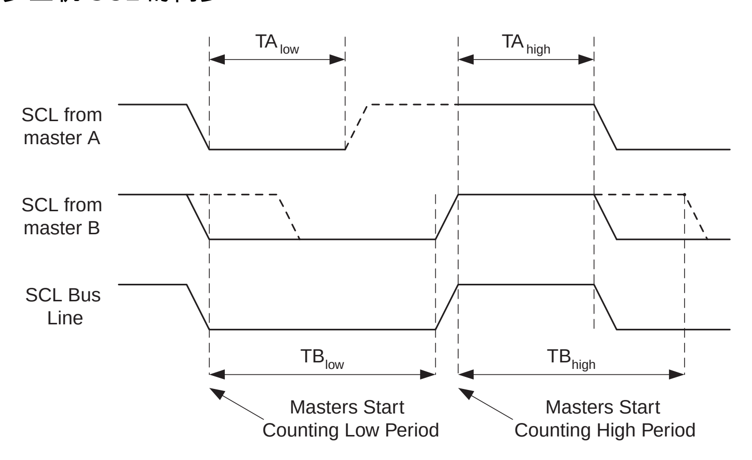
多主机总线系统，仲裁和同步

TWI 协议允许总线上由多个主机。特别要注意的是即使有多个主机同时开始发生数据，也要保证发送正常进行。多主机系统中有两个问题：

- 算法必须只能允许一个主机完成传送。当其余主机发现它们失去选择权后应停止传送。这个选择过程称为仲裁。当竞争中的主机发现其仲裁失败，应立即转换到从机模式检测是否被获得总线控制权的的主机寻址。事实上多主机同时传送时不应该让从机检测到，即不许破坏数据在总线上的传送。
- 不同的主机可能使用不同的 SCL 频率。为保证传送的一致性，必须设计一种同步主机时钟的方案。这会简化仲裁过程。

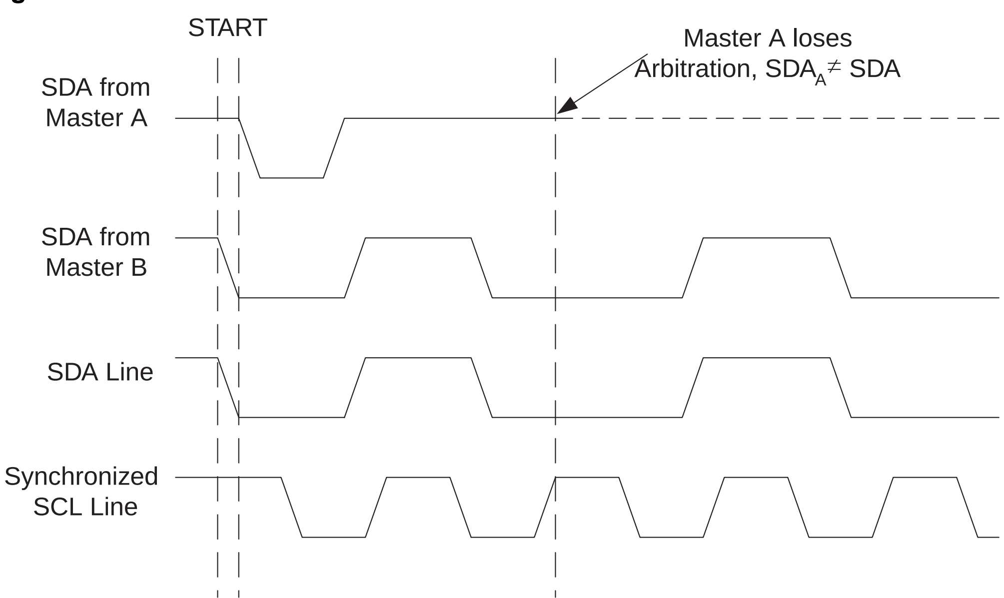
总线的线与功能用来解决上述问题。将所有的主机时钟进行与操作，会生成组合的时钟，其高电平时间等于所有主机中最短的一个；低电平时间则等于所有主机中最长的一个。所有的主机都监听 SCL，使其可以有效地计算本身高 / 低电平与组合 SCL 信号高 / 低电平的时间差异。

Figure 92. 多主机 SCL 的同步



输出数据之后所有的主机都持续监听 SDA 来实现仲裁。如果从 SDA 读回的数值与主机输出的数值不匹配，该主机即失去仲裁。要注意只有当一个主机输出高电平的 SDA，而其它主机输出为低，该主机才会失去仲裁，并立即转为从机模式，检测是否被胜出的主机寻址。失去仲裁的主机必须将 SDA 置高，但在当前的数据或地址包结束之前还可以产生时钟信号。仲裁将会持续到系统只有一个主机。这可能会占用许多比特。如果几个主机对相同的从机寻址，仲裁将会持续到数据包。

Figure 93. 两主机之间的仲裁



注意不允许在以下情况进行仲裁：

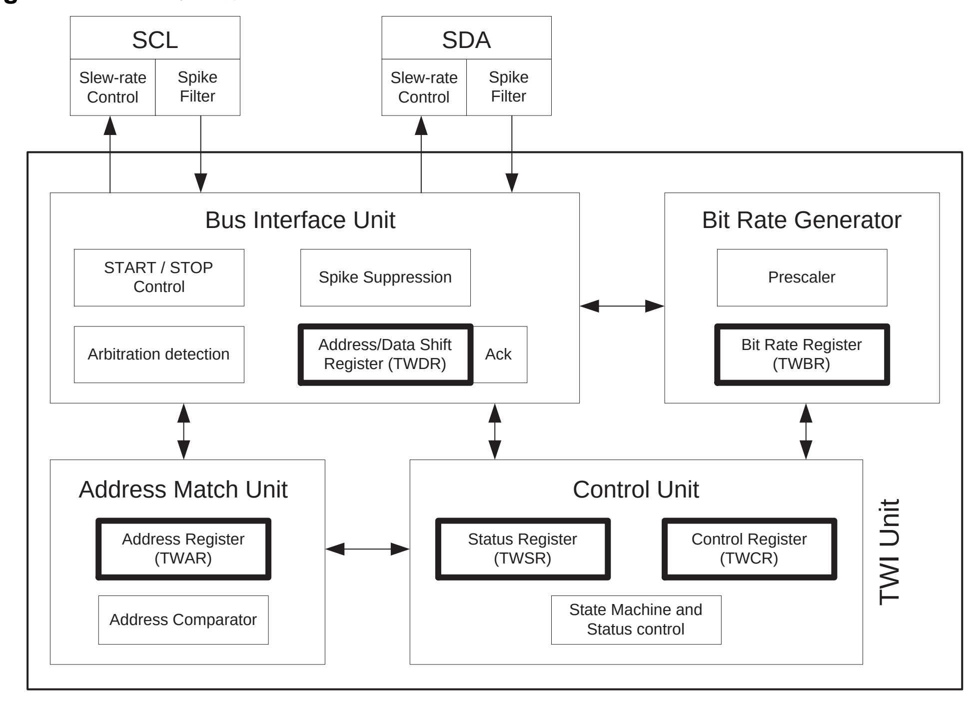
- 一个 REPEATED START 状态与一个数据位。
- 一个 STOP 状态与一个数据位。
- 一个 REPEATED START 状态与一个 STOP 状态。

应用软件应考虑上述情况，保证不会出现这些非法仲裁状态。这意味着在多主机系统中，所有的数据传输必须由相同的 SLA+R/W 与数据包组合组成。换句话说：所有的传送必须包含相同数目的数据包，否则仲裁结果无法定义。

TWI 模块综述

TWI模块由几个子模块组成，如Figure 94所示。所有位于粗线之中的寄存器可以通过AVR数据总线进行访问。

Figure 94. TWI 模块概述



Scl 和 SDA 引脚

SCL 与SDA为MCU的 TWI接口引脚。引脚的输出驱动器包含一个波形斜率限制器以满足TWI 规范。引脚的输入部分包括尖峰抑制单元以去除小于 50 ns 的毛刺。当相应的端口设置为 SCL 与 SDA 引脚时，可以使能 I/O 口内部的上拉电阻，这样可省掉外部的上拉电阻。

比特率发生器单元

TWI工作于主机模式时，比特率发生器控制时钟信号SCL的周期。具体由TWI状态寄存器TWSR的预分频系数以及比特率寄存器TWBR设定。当TWI工作在从机模式时，不需要对比特率或预分频进行设定，但从机的 CPU 时钟频率必须大于 TWI 时钟线 SCL 频率的 16 倍。注意，从机可能会延长 SCL 低电平的时间，从而降低 TWI 总线的平均时钟周期。SCL 的频率根据以下的公式产生：

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

- TWBR = TWI 比特率寄存器的数值
- TWPS = TWI 状态寄存器预分频的数值

Note: TWI 工作在主机模式时，TWBR 值应该不小于 10。否则主机会在 SDA 与 SCL 产生错误输出作为提示信号。问题出现于 TWI 工作在主机模式下，向从机发送 Start + SLA + R/W 的时候 (不需要真的有从机与总线连接)。

总线接口单元

该单元包括数据与地址移位寄存器 TWDR，START/STOP 控制器和总线仲裁判定硬件电路。 TWDR 寄存器用于存放发送或接收的数据或地址。除了 8 位的 TWDR，总线接口单元还有一个寄存器，包含了用于发送或接收应答的 (N)ACK。这个 (N)ACK 寄存器不能由程序直接访问。当接收数据时，它可以通过 TWI 控制寄存器 TWCR 来置位或清零；在发送数据时， (N)ACK 值由 TWCR 的设置决定。

START/STOP 控制器负责产生和检测 TWI 总线上的 START、REPEATED START 与 STOP 状态。即使在 MCU 处于休眠状态时，START/STOP 控制器仍然能够检测 TWI 总线

上的 START/STOP 条件，当检测到自己被 TWI 总线上的主机寻址时，将 MCU 从休眠状态唤醒。

如果 TWI 以主机模式启动了数据传输，仲裁检测电路将持续监听总线，以确定是否可以通过仲裁获得总线控制权。如果总线仲裁单元检测到自己在总线仲裁中丢失了总线控制权，则通知 TWI 控制单元执行正确的动作，并产生合适的状态码。

地址匹配单元

地址匹配单元将检测从总线上接收到的地址是否与 TWAR 寄存器中的 7 位地址相匹配。如果 TWAR 寄存器的 TWI 广播应答识别使能位 TWGCE 为 "1"，从总线接收到的地址也会与广播地址进行比较。一旦地址匹配成功，控制单元将得到通知以进行正确地响应。TWI 可以响应，也可以不响应主机的寻址，这取决于 TWCR 寄存器的设置。即使 MCU 处于休眠状态时，地址匹配单元仍可继续工作。一旦主机寻址到这个器件，就可以将 MCU 从休眠状态唤醒。在 TWI 由于地址匹配将 MCU 从掉电状态唤醒期间，如有其他中断发生，TWI 将放弃操作，并返回到空闲状态。如果这会引起其他问题，那么在进入掉电休眠模式之前需要确保只有 TWI 地址匹配中断被使能。

控制单元

控制单元监听 TWI 总线，并根据 TWI 控制寄存器 TWCR 的设置作出相应的响应。当 TWI 总线上产生需要应用程序干预处理的事件时，TWI 中断标志位 TWINT 置位。在下一个时钟周期，TWI 状态寄存器 TWSR 被表示这个事件的状态码字所更新。在其它时间里，TWSR 的内容为一个表示无事件发生的特殊状态字。一旦 TWINT 标志位置 "1"，时钟线 SCL 即被拉低，暂停 TWI 总线上的数据传输，让用户程序处理事件。

在下列状况出现时， TWINT 标志位置位：

- 在 TWI 传送完 START/REPEATED START 信号之后。
- 在 TWI 传送完 SLA+R/W 数据之后。
- 在 TWI 传送完地址字节之后。
- 在 TWI 总线仲裁失败之后。
- 在 TWI 被主机寻址之后 (广播方式或从机地址匹配)。
- 在 TWI 接收到一个数据字节之后。
- 作为从机工作时， TWI 接收到 STOP 或 REPEATED START 信号之后。
- 由于非法的 START 或 STOP 信号造成总线错误时。



TWI 寄存器说明

TWI 比特率寄存器 - TWBR

Bit	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bits 7..0 – TWI 比特率寄存器**

TWBR 为比特率发生器分频因子。比特率发生器是一个分频器，在主机模式下产生SCL时钟频率。比特率计算公式请见 P 186“ 比特率发生器单元 ” 。

TWI 控制寄存器 - TWCR

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
读 / 写	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
初始值	0	0	0	0	0	0	0	0	

TWCR 用来控制TWI操作。它用来使能TWI，通过施加START到总线上来启动主机访问，产生接收器应答，产生 STOP 状态，以及在写入数据到 TWDR 寄存器时控制总线的暂停等。这个寄存器还可以给出在 TWDR 无法访问期间，试图将数据写入到 TWDR 而引起的写入冲突信息。

- **Bit 7 – TWINT: TWI 中断标志**

当 TWI 完成当前工作，希望应用程序介入时 TWINT 置位。若 SREG 的 I 标志以及 TWCR 寄存器的 TWIE 标志也置位，则 MCU 执行 TWI 中断例程。当 TWINT 置位时， SCL 信号的低电平被延长。 TWINT 标志的清零必须通过软件写 "1" 来完成。执行中断时硬件不会自动将其改写为 "0"。要注意的是，只要这一位被清零， TWI 立即开始工作。因此，在清零 TWINT 之前一定要首先完成对地址寄存器 TWAR，状态寄存器 TWSR，以及数据寄存器 TWDR 的访问。

- **Bit 6 – TWEA: 使能 TWI 应答**

TWEA 标志控制应答脉冲的产生。若 TWEA 置位，出现如下条件时接口发出 ACK 脉冲：

1. 器件的从机地址与主机发出的地址相符合。
2. TWAR 的 TWGCE 置位时接收到广播呼叫。
3. 在主机 / 从机接收模式下接收到一个字节的数据。

将 TWEA 清零可以使器件暂时脱离总线。置位后器件重新恢复地址识别。

- **Bit 5 – TWSTA: TWI START 状态位**

当 CPU 希望自己成为总线上的主机时需要置位 TWSTA。 TWI 硬件检测总线是否可用。若总线空闲，接口就在总线上产生 START 状态。若总线忙，接口就一直等待，直到检测到一个 STOP 状态 ，然后产生 START 以声明自己希望成为主机。发送 START 之后软件必须清零 TWSTA。

- **Bit 4 – TWSTO: TWI STOP 状态位**

在主机模式下，如果置位 TWSTO，TWI 接口将在总线上产生 STOP 状态，然后 TWSTO 自动清零。在从机模式下，置位 TWSTO 可以使接口从错误状态恢复到未被寻址的状态。此时总线上不会有 STOP 状态产生，但 TWI 返回一个定义好的未被寻址的从机模式且释放 SCL 与 SDA 为高阻态。

- **Bit 3 – TWWC: TWI 写冲突标志**

当 TWINT 为低时写数据寄存器 TWDR 将置位 TWWC。每一次对 TWDR 的写访问都将更新此标志。

• **Bit 2 – TWEN: TWI 使能位**

TWEN 位用于使能TWI操作与激活TWI接口。当TWEN位被写为"1"时，TWI引脚将I/O引脚切换到 SCL 与 SDA 引脚，使能波形斜率限制器与尖峰滤波器。如果该位清零， TWI 接口模块将被关闭，所有 TWI 传输将被终止。

• **Bit 1 – Res: 保留**

保留，读返回值为 ”0”。

• **Bit 0 – TWIE: TWI 中断使能**

当 SREG 的 I 以及 TWIE 置位时，只要 TWINT 为 "1"， TWI 中断就激活。

TWI 状态寄存器 - TWSR

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
读 / 写	R	R	R	R	R	R	R/W	R/W	
初始值	1	1	1	1	1	0	0	0	

• **Bits 7..3 – TWS: TWI 状态**

这 5 位用来反映 TWI 逻辑和总线的状态。不同的状态代码将会在后面的部分描述。注意从 TWSR 读出的值包括 5 位状态值与 2 位预分频值。检测状态位时设计者应屏蔽预分频位为 "0”。这使状态检测独立于预分频器设置。在无特殊声明的情况下，在手册中使用该方法。

• **Bit 2 – Res: 保留**

保留，读返回值为 "0”。

• **Bits 1..0 – TWPS: TWI 预分频位**

这两位可读 / 写，用于控制比特率预分频因子。

Table 87. TWI 位速率预分频因子

TWPS1	TWPS0	预分频因子值
0	0	1
0	1	4
1	0	16
1	1	64

如何计算比特率请见 P 186“ 比特率发生器单元 ” 。 TWPS1..0 值用于公式中。

TWI 数据寄存器 - TWDR

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	1	1	1	1	1	1	1	1	

在发送模式， TWDR 包含了要发送的字节；在接收模式， TWDR 包含了接收到的数据。当 TWI 接口没有进行移位工作 (TWINT 置位) 时这个寄存器是可写的。在第一次中断发生之前用户不能够初始化数据寄存器。只要 TWINT 置位，TWDR 的数据就是稳定的。在数据移出时，总线上的数据同时移入寄存器。 TWDR 总是包含了总线上出现的最后一个字节，除非 MCU 是从掉电或省电模式被 TWI 中断唤醒。此时 TWDR 的内容没有定义。总线仲裁失败时，主机将切换为从机，但总线上出现的数据不会丢失。 ACK 的处理由 TWI 逻辑自动管理， CPU 不能直接访问 ACK。

• **Bits 7..0 – TWD: TWI 数据寄存器**



根据状态的不同，其内容为要发送的下一个字节，或是接收到的数据。

TWI(从机) 地址寄存器 - TWAR

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	1	1	1	1	1	1	1	0	

TWAR 的高 7 位为从机地址。工作于从机模式时 , TWI 将根据这个地址进行响应。主机模式不需要此地址。在多主机系统中， TWAR 需要进行设置以便其他主机访问自己。

TWAR 的 LSB 用于识别广播地址 (0x00)。器件内有一个地址比较器。一旦接收到的地址和本机地址一致，芯片就请求中断。

- **Bits 7..1 – TWA: TWI 从机地址寄存器**

其值为从机地址。

- **Bit 0 – TWGCE: 使能 TWI 广播识别**

置位后 MCU 可以识别 TWI 总线广播。

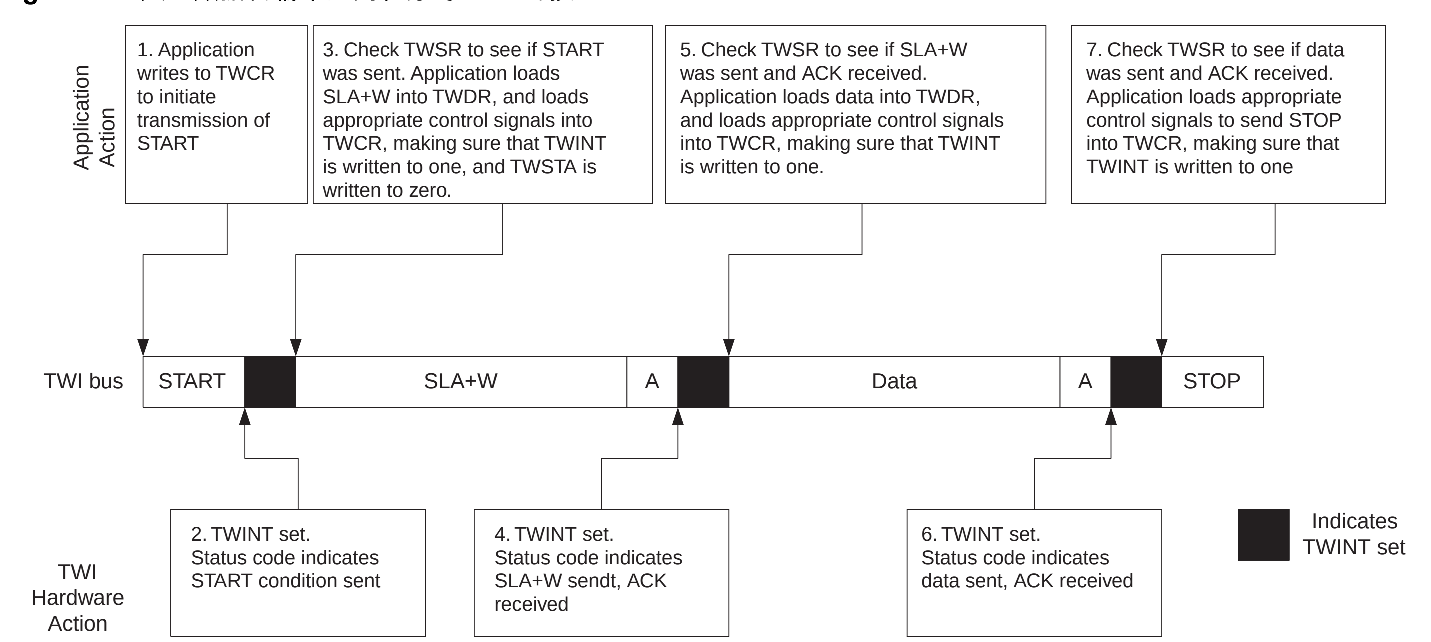
使用 TWI

AVR的 TWI接口是面向字节和基于中断的。所有的总线事件 , 如接收到一个字节或发送了一个 START 信号等，都会产生一个 TWI 中断。由于 TWI 接口是基于中断的，因此 TWI 接口在字节发送和接收过程中，不需要应用程序的干预。 TWCR 寄存器的 TWI 中断允许 TWIE位和SREG寄存器的全局中断允许位一起决定了应用程序是否响应TWINT标志位产生的中断请求。如果 TWIE 被清零，应用程序只能采用轮询 TWINT 标志位的方法来检测 TWI 总线状态。

当 TWINT 标志位置 "1" 时，表示 TWI 接口完成了当前的操作，等待应用程序的响应。在这种情况下，TWI 状态寄存器 TWSR 包含了表明当前 TWI 总线状态的值。应用程序可以读取 TWCR 的状态码 , 判别此时的状态是否正确 , 并通过设置 TWCR 与 TWDR 寄存器，决定在下一个 TWI 总线周期 TWI 接口应该如何工作。

Figure 95 给出应用程序与 TWI 接口连接的例子。该例中 , 主机发送一个数据字节给从机。这里只是简述，本节的后面会有更多的解释，还有简单的代码例程。

Figure 95. 典型数据传输中应用程序与 TWI 的接口



1. TWI 传输的第一步是发送START信号。通过对 TWCR写入特定值 ,指示TWI硬件发送 START 信号。写入的值将在后面说明。在写入值时 TWINT 位要置位，这非常重要。给 TWINT 写 "1" 清除此标志。 TWCR 寄存器的 TWINT 置位期间 TWI 不会启动任何操作。一旦 TWINT 清零， TWI 由 START 信号启动数据传输。
2. START信号被发送后 ,TWCR寄存器的 TWINT标志位置位，TWCR更新为新的状态码，表示 START 信号成功发送。
3. 应用程序应检验 TWSR ,确定 START 信号已成功发送。如果 TWSR 显示为其它，应用程序可以执行一些指定操作，比如调用错误处理程序。如果状态码与预期一致，应用程序必须将 SLA+W 载入 TWDR。TWDR 可同时在地址与数据中使用。TWDR 载入 SLA+W 后 ,TWCR 必须写入特定值指示 TWI 硬件发送 SLA+W 信号。写入的值将在后面说明。在写入值时 TWINT 位要置位，这非常重要。给 TWINT 写 "1" 清除此标志。 TWCR 寄存器的 TWINT 置位期间 TWI 不会启动任何操作。一旦 TWINT 清零， TWI 启动地址包的传送。
4. 地址包发送后 ,TWCR 寄存器的 TWINT 标志位置位，TWDR 更新为新的状态码，表示地址包成功发送。状态代码还会反映从机是否响应包。
5. 应用程序应检验 TWSR , 确定地址包已成功发送、ACK 为期望值。如果 TWSR 显示为其它，应用程序可能执行一些指定操作，比如调用错误处理程序。如果状态码与预期一致，应用程序必须将数据包载入 TWDR。随后， TWCR 必须写入特定值指示 TWI 硬件发送 TWDR 中的数据包。写入的值将在后面说明。在写入值时 TWINT 位要置位，这非常重要。 TWCR 寄存器中的 TWINT 置位期间 TWI 不会启动任何操作。一旦 TWINT 清零， TWI 启动数据包的传输。
6. 数据包发送后 ,TWCR 寄存器的 TWINT 标志位置位，TWSR 更新为新的状态码，表示数据包成功发送。状态代码还会反映从机是否响应包。
7. 应用程序应检验 TWSR ,确定地址包已成功发送、ACK 为期望值。如果 TWSR 显示为其它，应用程序可能执行一些指定操作，比如调用错误处理程序。如果状态码与预期一致， TWCR 必须写入特定值指示 TWI 硬件发送 STOP 信号。写入的值将在后面说明。在写入值时 TWINT 位要置位，这非常重要。给 TWINT 写 "1" 清除此标志。 TWCR 寄存器中的 TWINT 置位期间 TWI 不会启动任何操作。一旦 TWINT清零 ,TWI启动STOP信号的传送。注意 TWINT在STOP状态发送后不会置位。



尽管示例比较简单，但它包含了 TWI 数据传输过程中的所有规则。总结如下：

- 当 TWI 完成一次操作并等待反馈时， TWINT 标志置位。直到 TWINT 清零，时钟线 SCL 才会拉低。
- TWINT标志置位时，用户必须用与下一个TWI总线周期相关的值更新TWI寄存器。例如， TWDR 寄存器必须载入下一个总线周期中要发送的值。
- 当所有的 TWI 寄存器得到更新，而且其它挂起的应用程序也已经结束， TWCR 被写入数据。写 TWCR 时， TWINT 位应置位。对 TWINT 写 "1" 清除此标志。 TWI 将开始执行由 TWCR 设定的操作。

下面给出了汇编与 C 语言例程。注意假设下面代码均已给出定义。

	汇编代码例子	C 例子	说明
1	**ldi** r16, (1<<TWINT)|(1<<TWSTA)| (1<<TWEN) **out** TWCR, r16	TWCR = (1<<TWINT)|(1<<TWSTA)| (1<<TWEN)	发出 START 信号
2	wait1: **in** r16,TWCR **sbrs** r16,TWINT **rjmp** wait1	**while** (!(TWCR & (1<<TWINT))) ;	等待 TWINT 置位， TWINT 置位表示 START 信号已发出
3	**in** r16,TWSR **andi** r16, 0xF8 **cpi** r16, START **brne** ERROR	**if** ((TWSR & 0xF8) != START) ERROR();	检验 TWI 状态寄存器，屏蔽预分频位，如果状态字不是 START 转出错处理
	ldi r16, SLA_W **out** TWDR, r16 **ldi** r16, (1<<TWINT) | (1<<TWEN) **out** TWCR, r16	TWDR = SLA_W; TWCR = (1<<TWINT) | (1<<TWEN);	将 SLA_W 载入 TWDR 寄存器， TWINT 位清零，启动发送地址
4	wait2: **in** r16,TWCR **sbrs** r16,TWINT **rjmp** wait2	**while** (!(TWCR & (1<<TWINT))) ;	等待 TWINT 置位， TWINT 置位表示总线命令 SLA+W 已发出，及收到应答信号 ACK/NACK
5	**in** r16,TWSR **andi** r16, 0xF8 **cpi** r16, MT_SLA_ACK **brne** ERROR	**if** ((TWSR & 0xF8) != MT_SLA_ACK) ERROR();	检验 TWI 状态寄存器，屏蔽预分频位，如果状态字不是 MT_SLA_ACK 转出错处理
	ldi r16, DATA **out** TWDR, r16 **ldi** r16, (1<<TWINT) | (1<<TWEN) **out** TWCR, r16	TWDR = DATA; TWCR = (1<<TWINT) | (1<<TWEN);	装入数据到 TWDR 寄存器， TWINT 清零，启动发送数据
6	wait3: **in** r16,TWCR **sbrs** r16,TWINT **rjmp** wait3	**while** (!(TWCR & (1<<TWINT))) ;	等待 TWINT 置位， TWINT 置位表示总线数据 DATA 已发送，及收到应答信号 ACK/NACK
7	**in** r16,TWSR **andi** r16, 0xF8 **cpi** r16, MT_DATA_ACK **brne** ERROR	**if** ((TWSR & 0xF8) != MT_DATA_ACK) ERROR();	检验 TWI 状态寄存器，屏蔽预分频器，如果状态字不是 MT_DATA_ACK 转出错处理
	ldi r16, (1<<TWINT)|(1<<TWEN)| (1<<TWSTO) **out** TWCR, r16	TWCR = (1<<TWINT)|(1<<TWEN)| (1<<TWSTO);	发送 STOP 信号

Note: 当 I/O 寄存器为扩展 I/O 寄存器时，必须用诸如 “LDS” 、“STS”、“SBRS”、“SBRC”、“SBR” 与 “CBR” 等可访问扩展 I/O 寄存器的指令代替 “IN”、 “OUT”、 “SBIS”、“SBIC”、 “CBI” 与 “SBI” 指令。





数据传输模式

TWI 可以工作于 4 个不同的模式 : 主机发送器 (MT)、主机接收器 (MR)、从机发送器 (ST) 及从机接收器 (SR)。同一应用程序可以使用几种模式。例如， TWI 可用 MT 模式给 TWI EEPROM 写入数据 ,用 MR 模式从 EEPROM 读取数据。如果系统中有其它主机存在 ，它们可能给 TWI 发送数据，此时就可以用 SR 模式。应用程序决定采用何种模式。

下面对每种模式进行具体说明。每种模式的状态码在详细说明数据发送的图中进行描述。这些图包含了如下的缩写 :

S：START 状态

Rs：REPEATED START 状态

R：读一个比特 (SDA 为高电平)

W：写一个比特 (SDA 为低电平)

A：应答位 (SDA 为低电平)

$\overline{A}$：无应答位 (SDA 为高电平)

Data：8 位数据

P：STOP 状态

SLA：从机地址

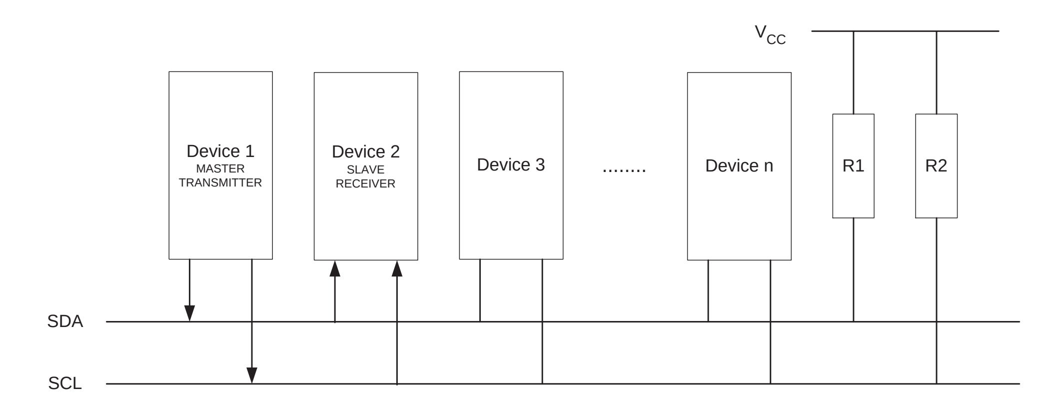
在 Figure 97 到 Figure 103 中，圆圈用来说明 TWINT 标志已经置位。圆圈中的数字用来表示预 TWSR 的数值，其中分频位屏蔽为 0。在这些地方应用程序必须执行合适的工作以继续 / 完成 TWI 传输。 TWI 传输被挂起，一直到 TWINT 标志被软件清零。

TWINT 标志置位后 ,TWSR 的状态码用来决定适当的软件操作。Table 88 到 Table 91 给出了每一个状态码所需的软件工作和后续串行传输的细节。注意在这些表中预分频位屏蔽为 0。

主机发送模式

在主机发送模式，主机可以向从机发送数据，如 Figure 96 所示。为进入主机模式，必须发送 START 信号。紧接着的地址包格式决定进入 MT 或 MR 模式。如果发送 SLA+W 进入 MT 模式；如果发送 SLA+R 则进入 MR 模式。本节所提到的状态字均假设其预分频位为 "0"。

Figure 96. 主机发送模式下的数据传输



通过在 TWCR 寄存器中写入下列数值发出 START 信号 :

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	1	0	X	1	0	X

TWEN必须置位以使能两线接口 ,TWSTA必须置"1"来发出START 信号且 TWINT必须置"1"来对 TWINT标志清零。TWI逻辑开始检测串行总线 ,一旦总线空闲就发送 START。接着中断标志 TWINT 置位， TWSR 的状态码为 $08 (见 Table 88)。为进入 MT 模式 ，必须发送 SLA+W。这可通过对 TWDR 写入 SLA+W 来实现。完成此操作后软件清零 TWINT标志， TWI 传输继续进行。这通过在 TWCR 寄存器中写入下述值完成：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	0	X	1	0	X

当SLA+W 发送完毕并接收到确认信号 ,主机的TWINT标志再次置位。此时主机的TWSR状态码可能是 $18、 $20 或 $38。对各状态码的正确响应列于 Table 88。

SLA+W 发送成功后可以开始发送数据包。这通过对 TWDR 写入数据实现。TWDR 只有在TWINT 为高时方可写入。否则，访问被忽略，寄存器 TWCR 的写碰撞位 TWWC 置位。TWDR更新后 ,TWINT位应清零来继续传送。这通过在 TWCR寄存器中写入下述值完成:

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	0	X	1	0	X

这过程会一直重复下去，直到最后的字节发送完且发送器产生 STOP 或 REPEATED START 信号 。STOP 信号通过在 TWCR 中写入下述值实现：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	1	X	1	0	X

REPEATED START 信号通过在 TWCR 中写入下述值实现：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	1	0	X	1	0	X

在 REPEATED START (状态 $10) 后，两线接口可以再次访问相同的从机，或不发送STOP 信号来访问新的从机。REPEATED START 使得主机可以在不丢失总线控制的条件下在从机、主机发送器及主机接收器模式间进行切换。

Table 88. 主机发送模式的状态码

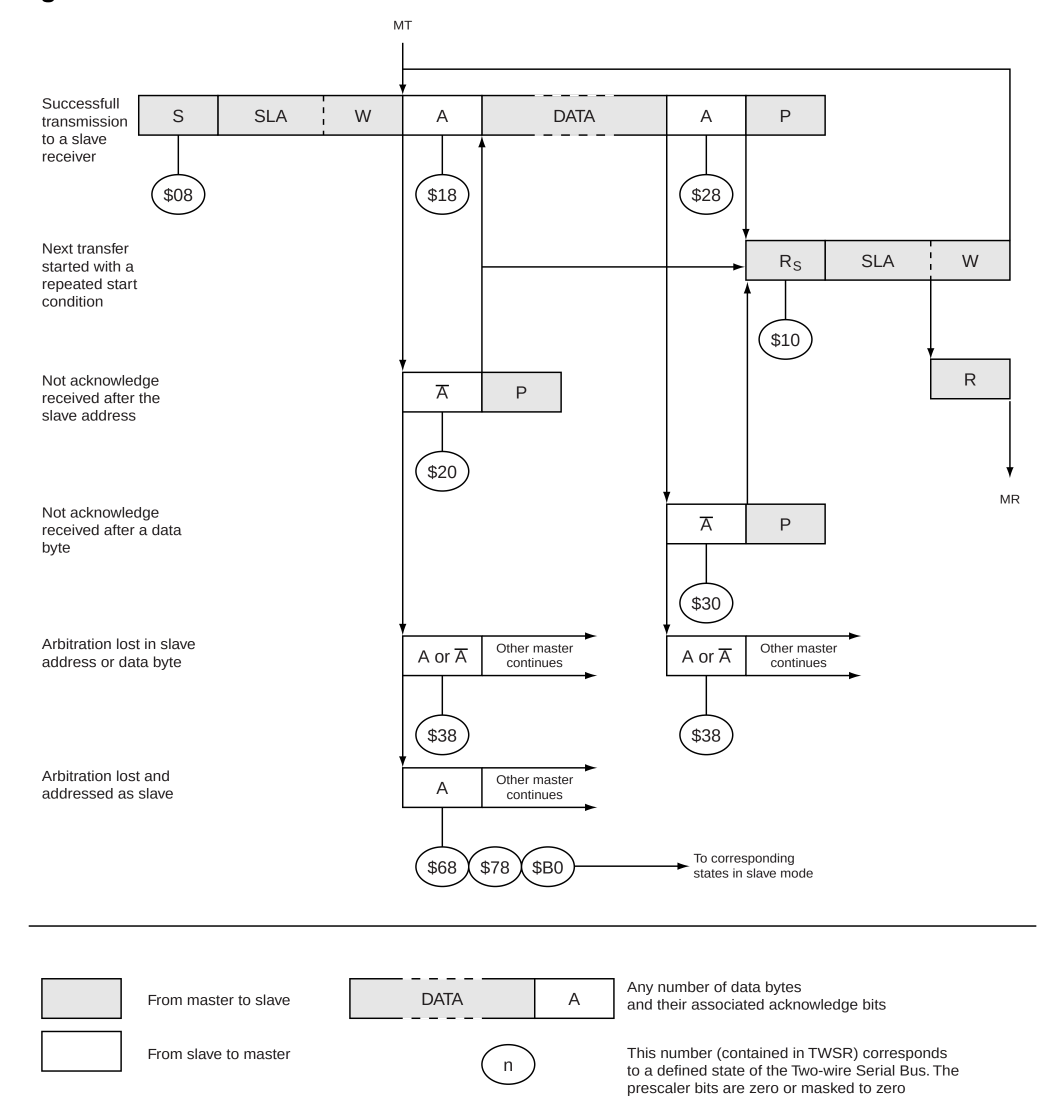
状态码 (TWSR) 预分频位为 "0"	2 线串行总线和 2 线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
		读 / 写 TWDR	对 TWCR 的操作				
			STA	STO	TWINT	TWEA	
$08	START 已发送	加载 SLA+W	0	0	1	X	将发送 SLA+W 将接收到 ACK 或 NOT ACK
$10	重复 START 已发送	加载 SLA+W 或 加载 SLA+R	0 0	0 0	1 1	X X	将发送 SLA+W 将接收到 ACK 或 NOT ACK 将发送 SLA+R 切换到主机接收模式
$18	SLA+W 已发送； 接收到 ACK	加载数据 (字节) 或 不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	将发送数据，接收 ACK 或 NOT ACK 将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位



Table 88. 主机发送模式的状态码

$20	SLA+W 已发送 接收到 NOT ACK	加载数据 (字节) 或 不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	将发送数据，接收 ACK 或 NOT ACK 将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位
$28	数据已发送 接收到 ACK	加载数据 (字节) 或 不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	将发送数据，接收 ACK 或 NOT ACK 将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位
$30	数据已发送 接收到 NOT ACK	加载数据 (字节) 或 不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	0 1 0 1	0 0 1 1	1 1 1 1	X X X X	将发送数据，接收 ACK 或 NOT ACK 将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位
$38	SLA+W 或数据的仲裁失败	不操作 TWDR 或 不操作 TWDR	0 1	0 0	1 1	X X	2 线串行总线将被释放，并进入未寻址从机模式 总线空闲后将发送 START

Figure 97. 主机发送模式的格式和状态

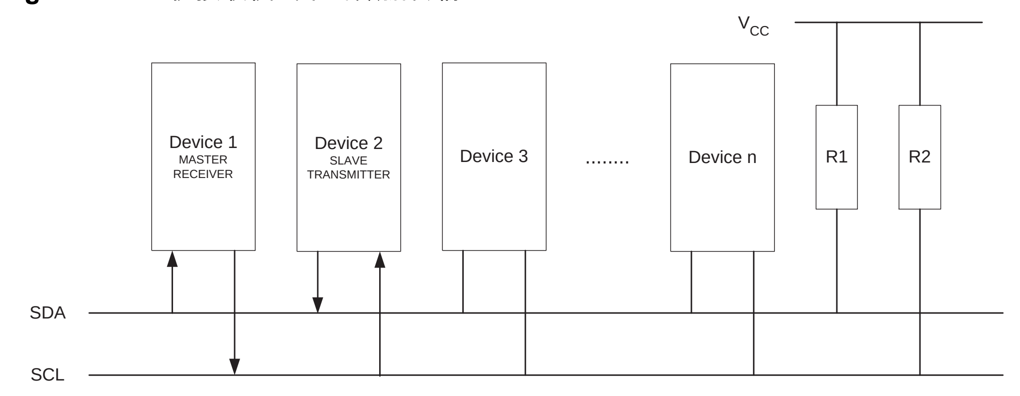




主机接收模式

在主机接收模式，主机可以从从机接收数据，如 Figure 98 所示。为进入主机模式，必须发送 START 信号。紧接着的地址包格式决定进入 MT 或 MR 模式。如果发送 SLA+W 进入 MT 模式；如果发送 SLA+R 则进入 MR 模式。本节所提到的状态字均假设其预分频位为 "0"。

Figure 98. 主机接收模式下的数据传输



通过在 TWCR 寄存器中写入下列数值发出 START 信号：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	1	0	X	1	0	X

TWEN必须置位以使能两线接口，TWSTA必须置"1"来发出START 信号且 TWINT必须置"1"来对TWINT标志清零。TWI逻辑开始检测串行总线，一旦总线空闲就发送START。接着中断标志 TWINT 置位， TWSR 的状态码为 $08 (见 Table 88)。为进入 MR 模式 ，必须发送 SLA+R。这可通过对 TWDR 写入 SLA+R 来实现。完成此操作后软件清零 TWINT标志， TWI 传输继续进行。这通过在 TWCR 寄存器中写入下述值完成：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	0	X	1	0	X

当 SLA+R 发送完毕并接收到确认信号，主机的 TWINT 标志再次置位。此时主机的 TWSR 状态码可能是 $38、 $40 或 $48。对各状态码的正确响应列于 Table 97。TWDR 只有在 TWINT 为高时才能读收到的数据。这过程会一直重复下去，直到最后的字节接收结束。接收完成后， MR 应通过在接收到最后的字节后发送 NACK 信号。发送器产生 STOP 或 REPEATED START 信号结束传送。STOP 信号通过在 TWCR 中写入下述值实现：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	0	1	X	1	0	X

REPEATED START 信号结束传送。STOP 信号通过在 TWCR 中写入下述值实现：

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	1	X	1	0	X	1	0	X

在 REPEATED START (状态 $10) 后，两线接口可以再次访问相同的从机，或不发送 STOP 信号来访问新的从机。REPEATED START 使得主机可以在不丢失总线控制的条件下在从机、主机发送器及主机接收器模式间进行切换。

Figure 99. 主机接收模式的格式和状态

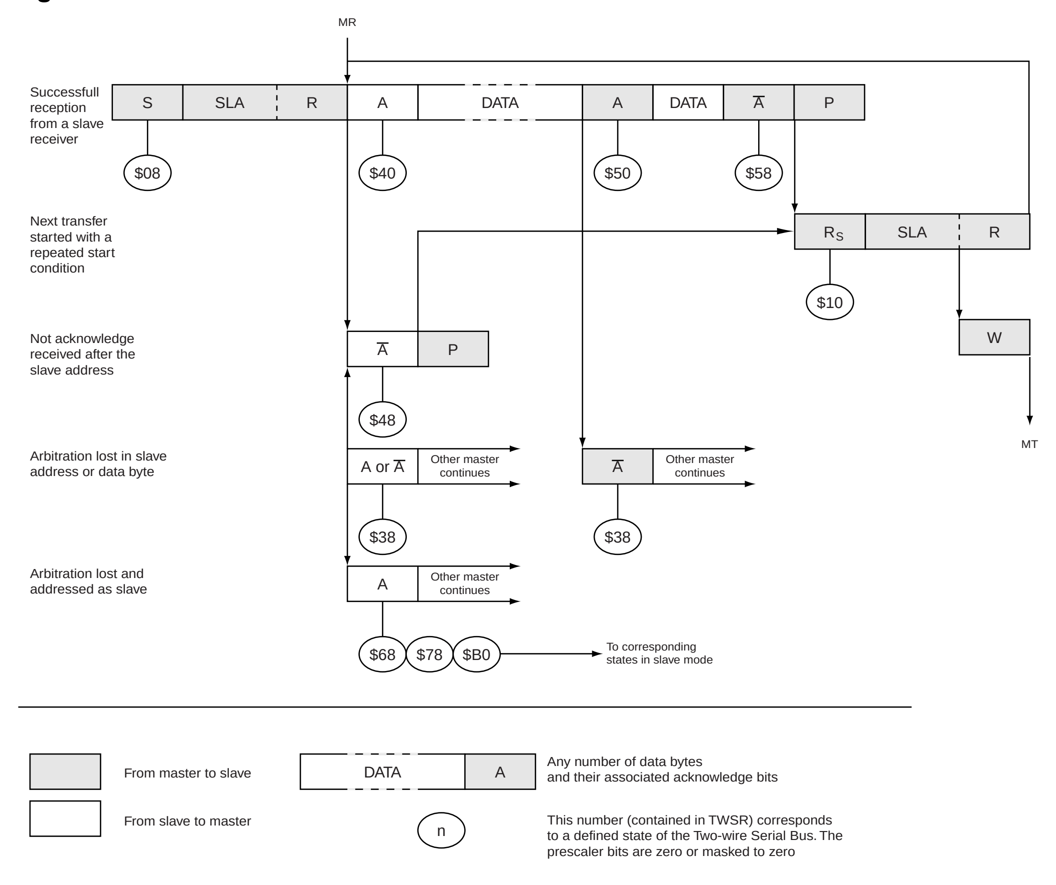


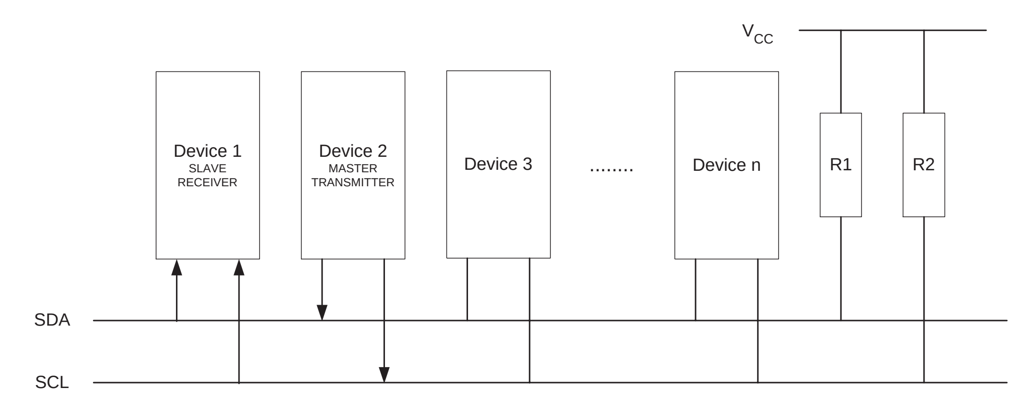
Table 89. 主机接收模式的状态码

状态码 (TWSR) 预分频位为 "0”	2 线串行总线和 2 线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
		读 / 写 TWDR	对 TWCR 的操作				
			STA	STO	TWINT	TWEA	
$08	START 已发送	加载 SLA+R	0	0	1	X	将发送 SLA+R 将接收到 ACK 或 NOT ACK
$10	重复 START 已发送	加载 SLA+R 或 加载 SLA+W	0 0	0 0	1 1	X X	将发送 SLA+R 将接收到 ACK 或 NOT ACK 将发送 SLA+W 逻辑切换到主机发送模式
$38	SLA+R 或 NOT ACK 的仲裁失败	不操作 TWDR 或 不操作 TWDR	0 1	0 0	1 1	X X	2 线串行总线将被释放，并进入未寻址从机模式 总线空闲后将发送 START
$40	SLA+R 已发送 接收到 ACK	不操作 TWDR 或 不操作 TWDR	0 0	0 0	1 1	0 1	接收数据，返回 NOT ACK 接收数据，返回 ACK
$48	SLA+R 已发送 接收到 NOT ACK	不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	1 0 1	0 1 1	1 1 1	X X X	将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位
$50	接收到数据 ACK 已返回	读数据或 读数据	0 0	0 0	1 1	0 1	接收数据，返回 NOT ACK 接收数据，返回 ACK
$58	接收到数据 NOT ACK 已返回	读数据或 读数据或 读数据	1 0 1	0 1 1	1 1 1	X X X	将发送重复 START 将发送 STOP， TWSTO 将复位 将发送 STOP，然后发送 START， TWSTO 将复位

从机接收模式

在从机接收模式，从机自主机接收数据，如 Figure 100 所示。本节所提到的状态字均假设其预分频位为 "0”。

Figure 100. 从机接收模式下的数据传输



为启动从机接收模式， TWAR 与 TWCR 设置如下：

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
值	器件本身从机地址							

前 7 位是主机寻址时从机响应的 TWI 接口地址。若 LSB 置位，则 TWI 接口响应广播地址 $00。否则忽略广播地址。

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	0	1	0	0	0	1	0	X

TWEN必须置位以使能TWI接口。TWEA也要置位以使主机寻址到自己(从机地址或广播)时返回确认信息 ACK。 TWSTA 和 TWSTO 必须清零。

初始化 TWAR 和 TWCR 之后， TWI 接口即开始等待，直到自己的从机地址 (或广播地址，如果 TWAR 的 TWGCE 置位的话) 出现在主机寻址地址当中，并且数据方向位为 0 (写)。然后 TWINT 标志置位， TWSR 则包含了相应的状态码。对各状态码的正确响应列于 Table 90。当 TWI 接口处于主机模式 (状态 $68 或 $78) 并发生仲裁失败时 CPU 将进入从机接收模式。

如果在传输过程中 TWEA 复位， TWI 接口在接收到下一个字节后将向 SDA 返回 " 无应答" 。 TWEA 复位时 TWI 接口不再响应自己的从机地址，但是会继续监视总线。一旦 TWEA置位就可以恢复地址识别和响应。也就是说，可以利用TWEA暂时将TWI接口从总线中隔离出来。

在除空闲模式外的其它休眠模式时， TWI 接口的时钟被关闭。若使能了从机接收模式，接口将利用总线时钟继续响应广播地址 / 从机地址。地址匹配将唤醒 CPU。在唤醒期间，TWI接口将保持SCL为低电平，直至TWCINT标志清零。当AVR时钟恢复正常运行后TWI可以接收更多的数据。显然如果 AVR 设置为长启动时间，时钟线 SCL 可能会长时间保持低，阻塞其它数据的传送。

当 MCU 从这些休眠模式唤醒时，和正常工作模式不同的是，数据寄存器 TWDR 的数据并不反映总线上出现的最后一个字节。

Table 90. 从机接收模式的状态码

状态码 (TWSR) 预分频位为 "0”	2线串行总线和2线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
		读 / 写 TWDR	对 TWCR 的操作				
			STA	STO	TWINT	TWEA	
$60	自己的 SLA+W 已经被接收 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
$68	SLA+R/W 作为主机的仲裁失败；自己的 SLA+W 已经被接收 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
$70	接收到广播地址 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
$78	SLA+R/W 作为主机的仲裁失败；接收到广播地址 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
$80	以前以自己的 SLA+W 被寻址；数据已经被接收 ACK 已返回	不操作 TWDR 或	X	0	1	0	接收数据，返回 NOT ACK
		不操作 TWDR	X	0	1	1	接收数据，返回 ACK
$88	以前以自己的 SLA+W 被寻址；数据已经被接收 NOT ACK 已返回	读数据或	0	0	1	0	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA
		读数据或	0	0	1	1	切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别
		读数据或	1	0	1	0	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START
		读数据	1	0	1	1	切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别；总线空闲时发送 START

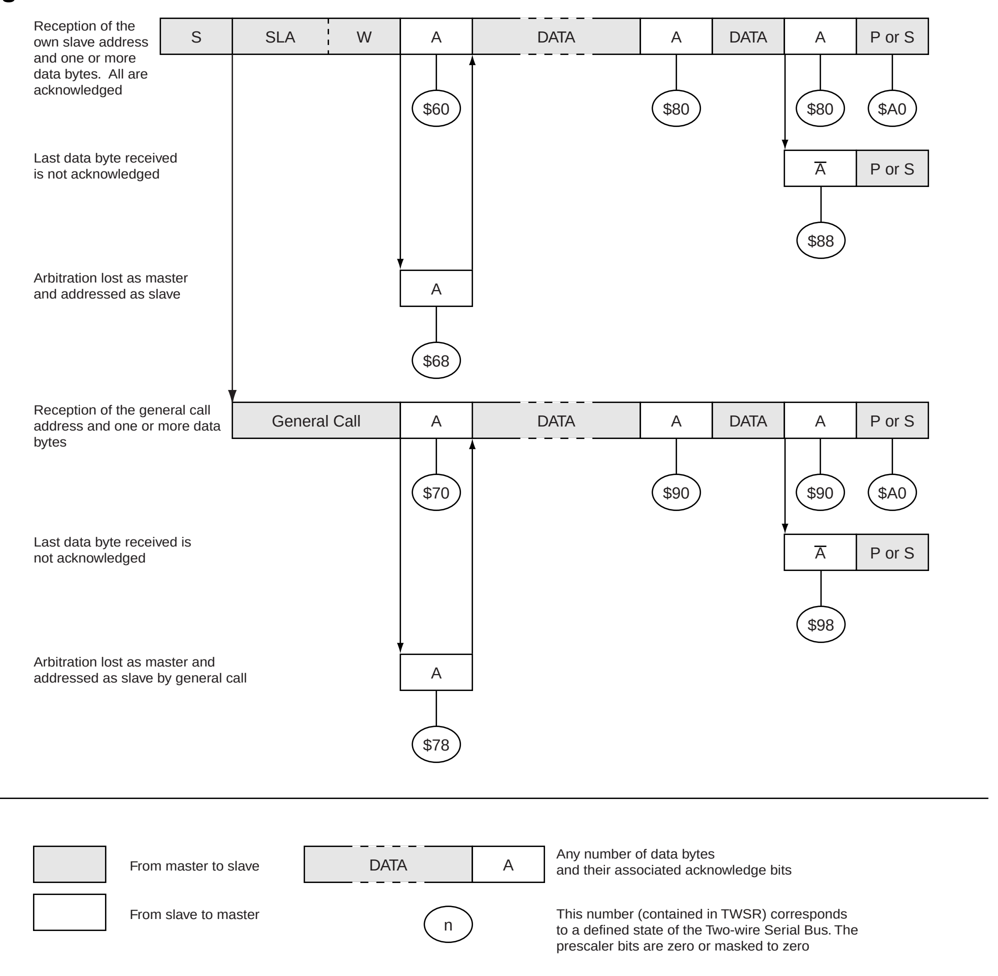




Table 90. 从机接收模式的状态码

$90	以前以广播方式被寻址；数据已经被接收 ACK 已返回	读数据或 读数据	X X	0 0	1 1	0 1	接收数据，返回 NOT ACK 接收数据，返回 ACK
$98	以前以广播方式被寻址；数据已经被接收 NOT ACK 已返回	读数据或 读数据或 r 读数据或 读数据	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA 切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别 切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START 切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别；总线空闲时发送 START
$A0	在以从机工作时接收到 STOP 或重复 START	读数据或 读数据或 r 读数据或 读数据	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA 切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别 切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START 切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别；总线空闲时发送 START

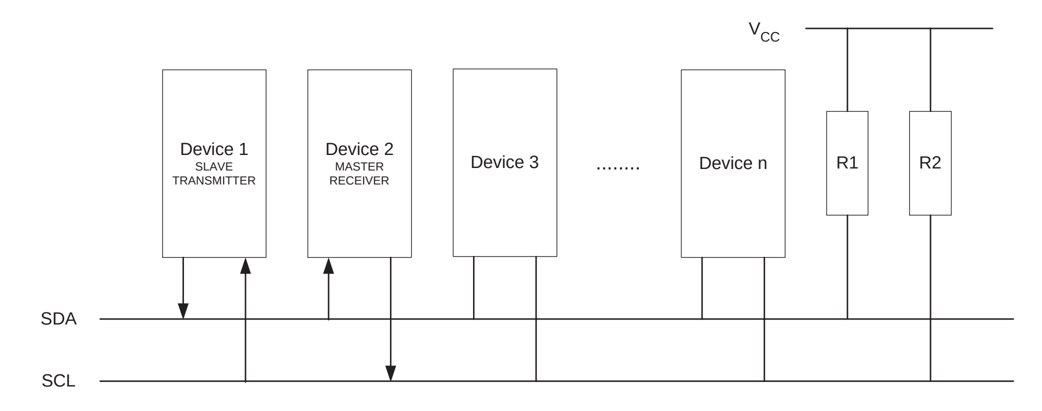
Figure 101. 从机接收模式的格式和状态



从机发送模式

在从机发送模式，从机可以向主机发送数据，如 Figure 102 所示。本节所提到的状态字均假设其预分频位为 "0"。

Figure 102. 从机发送模式下的数据传输



为启动从机发送模式， TWAR 与 TWCR 设置如下：

TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
值	器件本身从机地址							

前 7 位是主机寻址时从机响应的 TWI 接口地址。若 LSB 置位，则 TWI 接口响应广播地址 $00。否则忽略广播地址。

TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
值	0	1	0	0	0	1	0	X

TWEN必须置位以使能TWI接口。TWEA也要置位以便主机寻址到自己(从机地址或广播)时返回确认信息 ACK。 TWSTA 和 TWSTO 必须清零。

初始化 TWAR 和 TWCR 之后， TWI 接口即开始等待，直到自己的从机地址（或广播地址，如果 TWAR 的 TWGCE 置位的话）出现在主机寻址地址当中，并且数据方向位为 "1”(读)。然后 TWI 中断标志置位，TWSR 则包含了相应的状态码。对各状态码的正确响应列于 Table 91。当 TWI 接口处于主机模式 (状态 $B0) 并发生仲裁失败时 CPU 将进入从机发送模式。

如果在传输过程中 TWEA 复位，TWI 接口发送完数据之后进入状态 $C0 或 $C8。接口也切换到未寻址从机模式，忽略任何后续总线传输。从而主机接收到的数据全为 “1”。如果主机需要附加数据位 (通过发送 ACK)，即使从机已经传送结束，也进入状态 $C8。

TWEA复位时TWI接口不再响应自己的从机地址，但是会继续监视总线。一旦TWEA置位就可以恢复地址识别和响应。也就是说，可以利用 TWEA 暂时将 TWI 接口从总线中隔离出来。

在除空闲模式外的其它休眠模式时， TWI 接口的时钟被关闭。若使能了从机接收模式，接口将利用总线时钟继续响应广播地址 / 从机地址。地址匹配将唤醒 CPU。在唤醒期间，TWI接口将保持SCL为低电平，直至TWCINT标志清零。当AVR时钟恢复正常运行后可以发送更多的数据。显然如果 AVR 设置为长启动时间，时钟线 SCL 可能会长时间保持低，阻塞其它数据的传送。

当 MCU 从这些休眠模式唤醒时，和正常工作模式不同的是，数据寄存器 TWDR 的数据并不反映总线上出现的最后一个字节。

Table 91. 从机发送模式的状态码

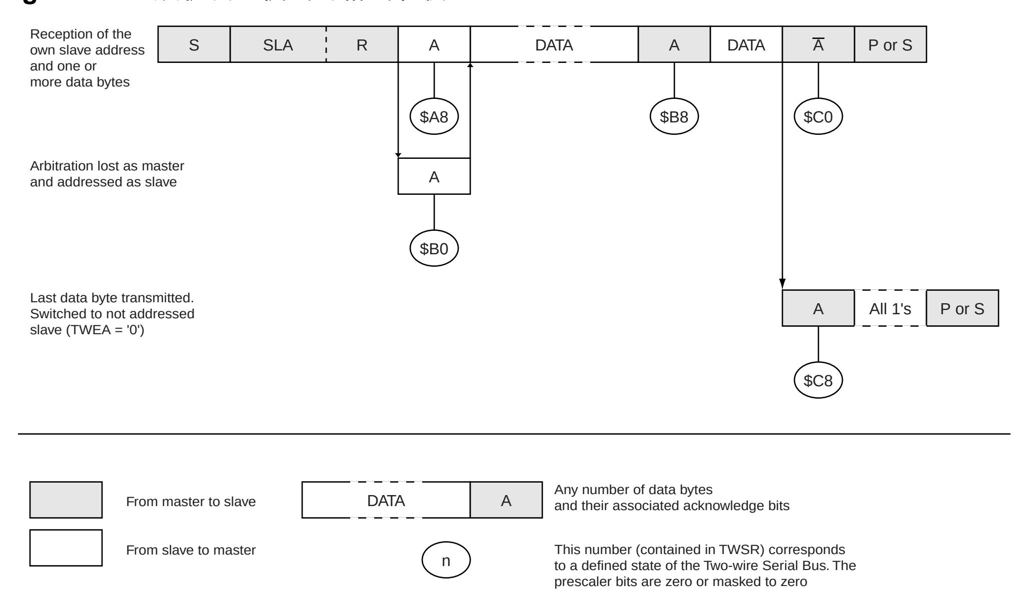
状态码 (TWSR) 预分频位为 "0”	2线串行总线和2线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
			对 TWCR 的操作				
		读 / 写 TWDR	STA	STO	TWINT	TWEA	

Table 91. 从机发送模式的状态码

$A8	自己的 SLA+R 已经被接收 ACK 已返回	加载一字节的数据或 加载一字节的数据	X X	0 0	1 1	0 1	发送一字节的数据，接收 NOT ACK 发送数据，接收 ACK
$B0	SLA+R/W 作为主机的仲裁失败；自己的 SLA+R 已经被接收 ACK 已返回	加载一字节的数据或 加载一字节的数据	X X	0 0	1 1	0 1	发送一字节的数据，接收 NOT ACK 发送数据，接收 ACK
$B8	TWDR 里数据已经发送 接收到 ACK	加载一字节的数据或 加载一字节的数据	X X	0 0	1 1	0 1	发送一字节的数据，接收 NOT ACK 发送数据，接收 ACK
$C0	TWDR 里数据已经发送 接收到 NOT ACK	不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA 切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别 切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START 切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别；总线空闲时发送 START
$C8	TWDR 的一字节数据已经发送 (TWAE = “0”); 接收到 ACK	不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR 或 不操作 TWDR	0 0 1 1	0 0 0 0	1 1 1 1	0 1 0 1	切换到未寻址从机模式；不再识别自己的 SLA 或 GCA 切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别 切换到未寻址从机模式；不再识别自己的 SLA 或 GCA ；总线空闲时发送 START 切换到未寻址从机模式；能够识别自己的 SLA ；若 GC = “1”， GCA 也可以识别；总线空闲时发送 START



Figure 103. 从机发送模式的格式和状态



其他状态

有两个状态码没有相应的 TWI 状态定义，见 Table 92。

状态 $F8 表明当前没有相关信息，因为中断标志 TWINT 为 "0"。这种状态可能发生在自己的 TWI 接口没有参与串行传输的时候。

状态 $00 表示在串行传输过程中发生了总线错误。当 START 或 STOP 出现在错误的位置时总线错误就会发生。比如说在地址和数据、地址和 ACK 之间出现了 START 或 STOP。总线错误将导致 TWINT 置位。为了从错误中恢复出来，必须置位标志 TWSTO，并通过写 "1" 以清零 TWINT。这将导致 TWI 接口进入未寻址从机模式、标志 TWSTO 被清零 (TWCR 的其他位不受影响)，以及 SDA 和 SCL 被释放，但是不会产生 STOP。

Table 92. 其它状态码

状态码 (TWSR) 预分频位为 "0"	2 线串行总线和 2 线串行硬件的状态	应用软件的响应					2 线串行硬件下一步应采取的动作
		读 / 写 TWDR	对 TWCR 的操作				
			STA	STO	TWINT	TWEA	
0xF8	没有相关的状态信息；TWINT = “0”	不操作 TWDR	不操作 TWCR				等待或进行当前传输
0x00	由于非法的 START 或 STOP 引起的总线错误	不操作 TWDR	0	1	1	X	只影响内部硬件；不会发送 STOP 到总线上。总线将释放并清零 TWSTO

将几个 TWI 模式组合到一起

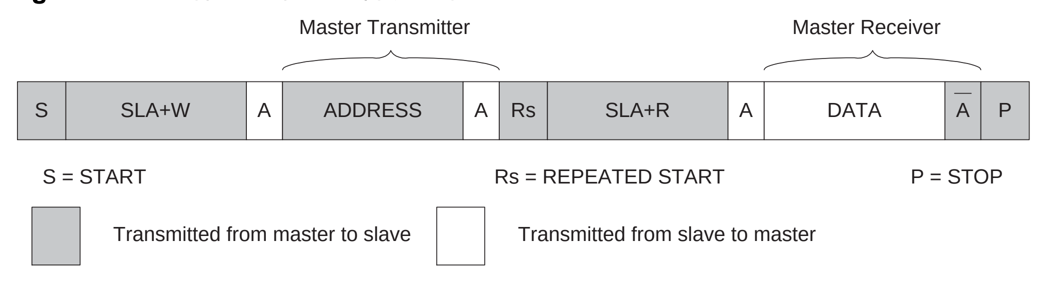
在某些情况下，为完成期望的工作，必须将几种TWI模式组合起来。例如从串行EEPROM 读取数据。典型的这种传输包括以下步骤：

1. 传输必须启动
2. 必须告诉 EEPROM 读取的位置
3. 必须完成读操作
4. 传送必须结束

注意数据可从主机传到从机，反之也可。首先主机必须告诉读从机读取实际的位置，因此需要使用 MT 模式；然后数据必须由从机读出，需要使用 MR 模式，但传送方向必须改变。在上述步骤中，主机必须保持对总线的控制，且以上各步骤应该自动进行。如果在多主机系统中违反这一规则，即在第二步与第三步之间其它主机改变 EEPROM 中的数据指

针，则主机读取的数据位置是错误的。传送方向改变是通过在发送地址字节与接收数据之间发送 REPEATED START 信号来实现的。在发送 REPEATED START 信号后，主机继续保持总线的控制权。下图给出传送的流程图。

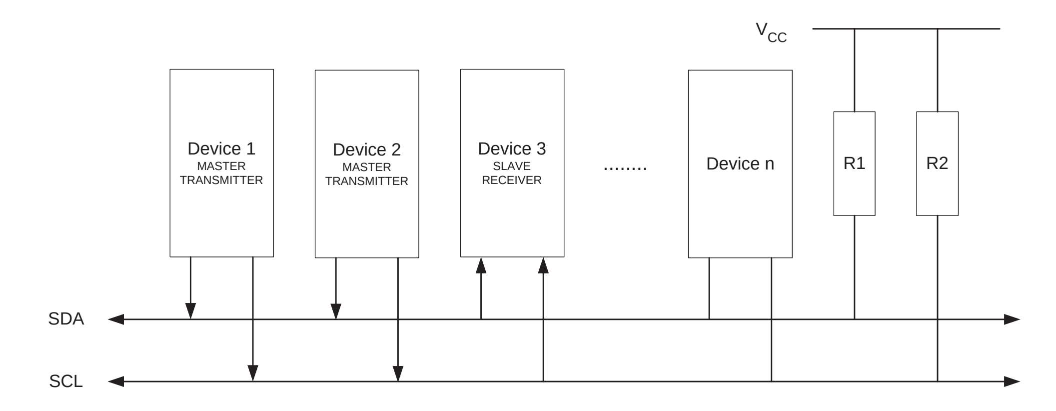
Figure 104. 几种 TWI 模式联合访问串行 EEPROM



多主机系统和仲裁

如果有多个主机连接在同一总线上，它们中的一个或多个也许会同时开始一个数据传送。TWI 协议确保在这种情况下，通过一个仲裁过程，允许其中的一个主机进行传送而不会丢失数据。总线仲裁的例子如下所述，该例中有两个主机试图向从接收器发送数据。

Figure 105. 仲裁示例



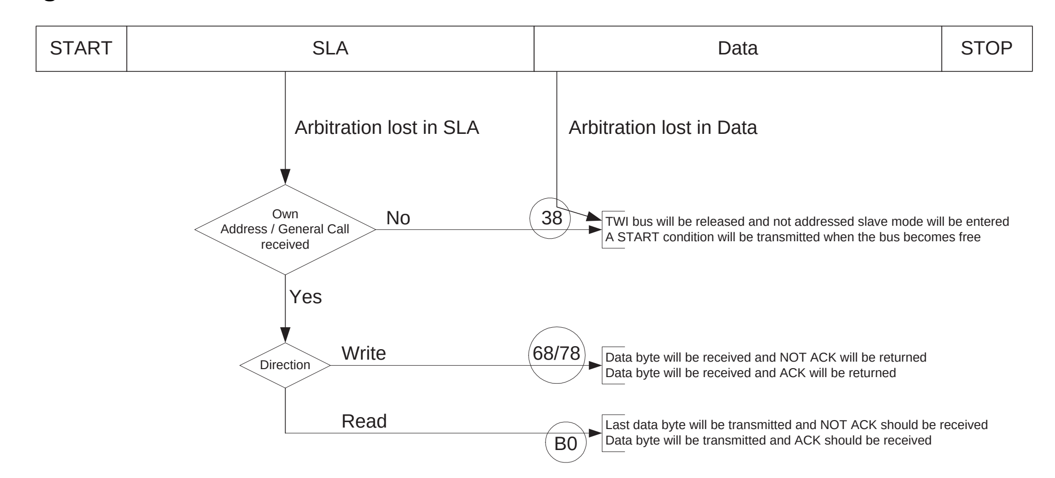
有几种不同的情况会产生总线仲裁过程：

- 两个或更多的主机同时与一个从机进行通信。在这种情况下，无论主机或从机都不知道有总线的竞争。
- 两个或更多的主机同时对同一个从机进行不同的数据或方向的访问。在这种情况下，会在 READ/WRITE 位或数据间发生仲裁。主机试图在 SDA 线上输出一个高电平时，如果其他主机已经输出 "0"，则该主机在总线仲裁中失败。失败的主机将转换成未被寻址的从机模式，或等待总线空闲后发送一个新的 START 信号，这由应用程序决定。
- 两个或更多的主机访问不同的从机。在这种情况下，总线仲裁在 SLA 发生。主机试图在 SDA 线上输出一个高电平时，如有其它主机已经输出 "0"，则该主机将在总线仲裁中失败。在 SLA 总线仲裁失败的主机将切换到从机模式，并检查自己是否被获得总线控制权的主机寻址。如果被寻址，它将进入 SR 或 ST 模式，这取决于 SLA 的 READ/WRITE 位的值。如果它未被寻址，将转换到未被寻址的从机模式或等待总线空闲，发送一个新的 START 信号，这由应用程序决定。

Figure 106 描述了总线仲裁的过程，图中的数字为 TWI 的状态值。



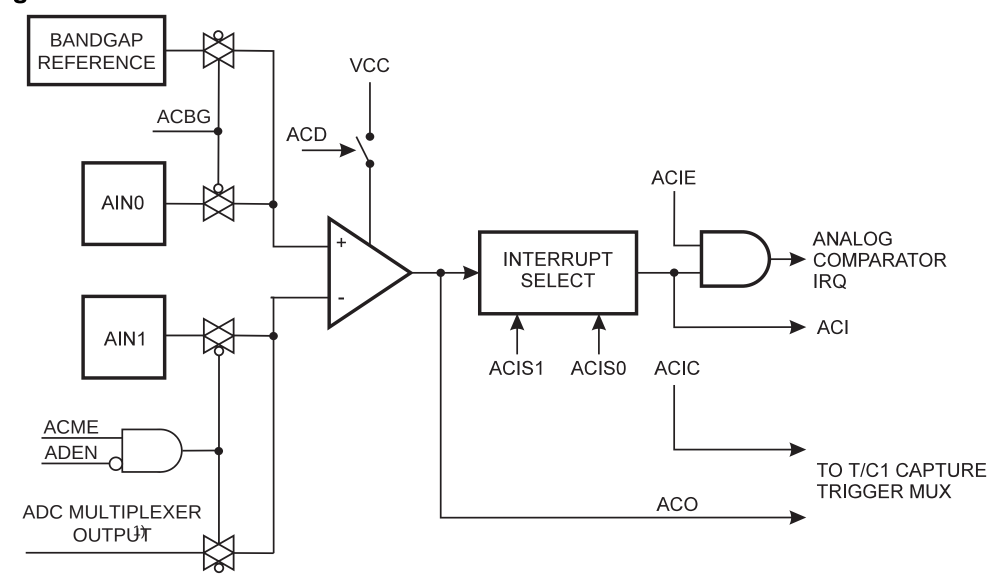
Figure 106. 总线仲裁过程



模拟比较器

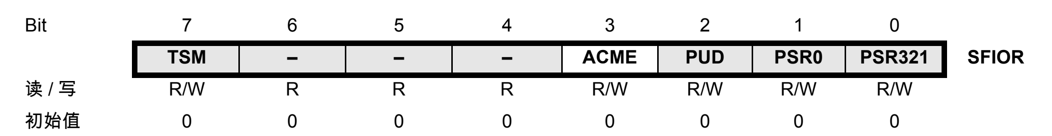
模拟比较器对正极 AIN0 的值与负极 AIN1 的值进行比较。当 AIN0 上的电压比负极 AIN1 上的电压要高时，模拟比较器的输出 ACO 即置位。比较器的输出可用来触发定时器 / 计数器 1 的输入捕捉功能。此外，比较器还可触发自己专有的、独立的中断。用户可以选择比较器是以上升沿、下降沿还是交替变化的边沿来触发中断。 Figure 107 为比较器及其外围逻辑电路的框图。

Figure 107. 模拟比较器框图



Notes: 1. 见 P 212Table 94 。
2. 模拟比较器的管脚分布见 P 2Figure 1 及 P 69Table 30 。

特殊功能 IO 寄存器 - SFIOR



Bit	7	6	5	4	3	2	1	0	
	TSM	–	–	–	ACME	PUD	PSR0	PSR321	SFIOR
读 / 写	R/W	R	R	R	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 3 – ACME: 模拟比较器多路复用器使能**

当此位为逻辑 "1” , 且 ADC 处于关闭状态 (ADCSRA 寄存器的 ADEN 为 "0”) 时 , ADC 多路复用器为模拟比较器选择负极输入。当此位为 "0” 时， AIN1 连接到比较器的负极输入端。更详细描述的请参见 P 211“ 模拟比较器多工输入 ” 。

模拟比较器控制和状态寄存器 - ACSR

Bit	7	6	5	4	3	2	1	0	
	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
读 / 写	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	N/A	0	0	0	0	0	

- **Bit 7 – ACD: 模拟比较器禁用**

ACD 置位时 , 模拟比较器的电源被切断。可以在任何时候设置此位来关掉模拟比较器。这可以减少器件工作模式及空闲模式下的功耗。改变 ACD 位时 , 必须清零 ACSR 寄存器的 ACIE 位来禁止模拟比较器中断。否则 ACD 改变时可能会产生中断。

- **Bit 6 – ACBG: 选择模拟比较器的能隙基准源**



ACBG 置位后，模拟比较器的正极输入由能隙基准源所取代。否则， AIN0 连接到模拟比较器的正极输入。见 P 50 “ 片内基准电压 ” 。

• **Bit 5 – ACO: 模拟比较器输出**

模拟比较器的输出经过同步后直接连到 ACO。同步机制引入了 1-2 个时钟周期的延时。

• **Bit 4 – ACI: 模拟比较器中断标志**

当比较器的输出事件触发了由 ACIS1 及 ACIS0 定义的中断模式时 ,ACI 置位。如果 ACIE 和 SREG 寄存器的全局中断标志 I 也置位 , 那么模拟比较器中断服务程序即得以执行 , 同时 ACI 被硬件清零。 ACI 也可以通过写 "1” 来清零。

• **Bit 3 – ACIE: 模拟比较器中断使能**

当 ACIE 位被置 "1” 且状态寄存器中的全局中断标志 I 也被置位时，模拟比较器中断被激活。否则中断被禁止。

• **Bit 2 – ACIC: 模拟比较器输入捕捉使能**

ACIC 置位后允许通过模拟比较器来触发 T/C1 的输入捕捉功能。此时比较器的输出被直接连接到输入捕捉的前端逻辑，从而使得比较器可以利用 T/C1 输入捕捉中断逻辑的噪声抑制器及触发沿选择功能。 ACIC 为 "0” 时模拟比较器及输入捕捉功能之间没有任何联系。为了使比较器可以触发 T/C1 的输入捕捉中断，定时器中断屏蔽寄存器 TIMSK 的 TICIE1 必须置位。

• **Bits 1, 0 – ACIS1, ACIS0: 模拟比较器中断模式选择**

这两位确定触发模拟比较器中断的事件。 Table 93 给出了不同的设置。

Table 93. ACIS1/ACIS0 设置

ACIS1	ACIS0	中断模式
0	0	比较器输出变化即可触发中断
0	1	保留
1	0	比较器输出的下降沿产生中断
1	1	比较器输出的上升沿产生中断

需要改变 ACIS1/ACIS0 时，必须清零 ACSR 寄存器的中断使能位来禁止模拟比较器中断。否则有可能在改变这两位时产生中断。

模拟比较器多工输入

可以选择 ADC7..0 之中的任意一个来代替模拟比较器的负极输入端。ADC 复用器可用来完成这个功能。当然，为了使用这个功能首先必须关掉 ADC。如果模拟比较器复用器使能位 (SFIOR 中的 ACME) 被置位 , 且 ADC 也已经关掉 (ADCSRA 寄存器的 ADEN 为 0)，则可以通过 ADMUX 寄存器的 MUX2..0 来选择替代模拟比较器负极输入的管脚，详见 Table 94。如果 ACME 清零或 ADEN 置位，则模拟比较器的负极输入为 AIN1。

Table 94. 模拟比较器复用输入

ACME	ADEN	MUX2..0	模拟比较器负极输入
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4

Table 94. 模拟比较器复用输入

ACME	ADEN	MUX2..0	模拟比较器负极输入
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

模数转换器

特点

- **10 位 精度**
- **0.5 LSB 的非线性度**
- **± 2 LSB 的绝对精度**
- **13 - 260 µs 的转换时间**
- **最高分辨率时采样率高达 15 kSPS**
- **8 路复用的单端输入通道**
- **7 路差分输入通道**
- **2 路可选增益为 10x 与 200x 的差分输入通道**
- **可选的左对齐 ADC 读数**
- **0 - V_{CC} 的 ADC 输入电压范围**
- **可选的 2.56V ADC 参考电压**
- **连续转换或单次转换模式**
- **ADC 转换结束中断**
- **基于睡眠模式的噪声抑制器**

ATmega128 有一个 10 位的逐次逼近型 ADC。ADC 与一个 8 通道的模拟多路复用器连接，能对来自端口 A 的 8 路单端输入电压进行采样。单端电压输入以 0V (GND) 为基准。

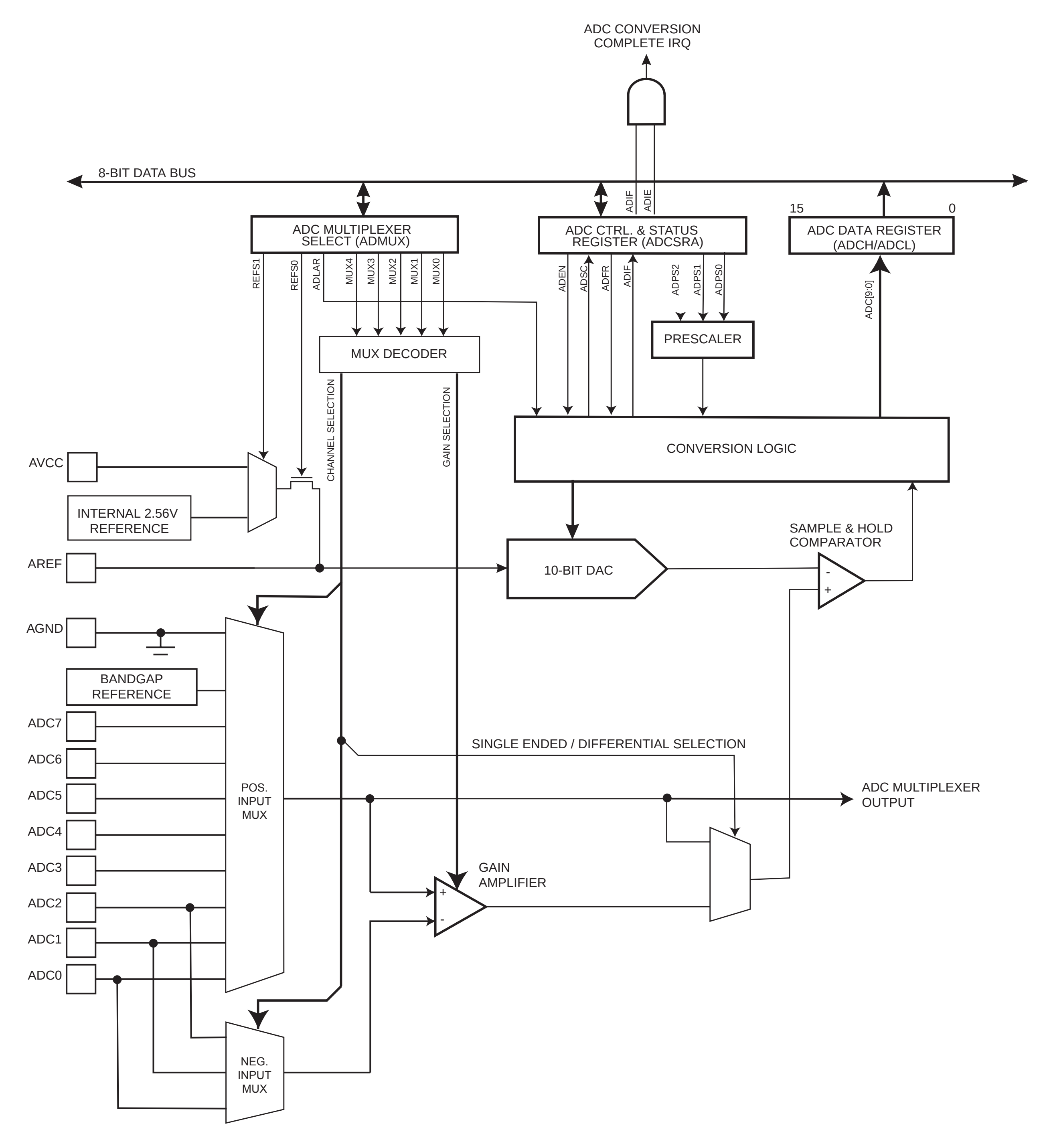
器件还支持 16 路差分电压输入组合。两路差分输入 (ADC1、 ADC0 与 ADC3、 ADC2) 有可编程增益级，在 A/D 转换前给差分输入电压提供 0 dB (1x)、 20 dB (10x) 或 46 dB (200x) 的放大级。七路差分模拟输入通道共享一个通用负端 (ADC1) , 而其他任何 ADC 输入可做为正输入端。如果使用 1x 或 10x 增益，可得到 8 位分辨率。如果使用 200x 增益，可得到 7 位分辨率。

ADC 包括一个采样保持电路 , 以确保在转换过程中输入到 ADC 的电压保持恒定。ADC 的框图如 Figure 108 所示。

ADC 由 AVCC 引脚单独提供电源。AVCC 与 V_{CC} 之间的偏差不能超过 ± 0.3V。请参考 P 219"ADC 噪声抑制器 " 来了解如何连接这个引脚。

标称值为 2.56V 的基准电压，以及 AVCC，都位于器件之内。基准电压可以通过在 AREF 引脚上加一个电容进行解耦，以更好地抑制噪声。

Figure 108. 模数转换器方框图



操作

ADC 通过逐次逼近的方法将输入的模拟电压转换成一个 10 位的数字量。最小值代表 GND，最大值代表 AREF 引脚上的电压再减去 1 LSB。通过写 ADMUX 寄存器的 REFSn 位可以把 AVCC 或内部 2.56V 的参考电压连接到 AREF 引脚。在 AREF 上外加电容可以对片内参考电压进行解耦以提高噪声抑制性能。

模拟输入通道与差分增益可以通过写 ADMUX 寄存器的 MUX 位来选择。任何 ADC 输入引脚，像 GND 及固定能隙参考电压，都可以作为 ADC 的单端输入。ADC 输入引脚可选做差分增益放大器的正或负输入。

如果选择差分通道，通过选择被选输入信号对的增益因子得到电压差分放大级。然后放大值成为 ADC 的模拟输入。如果使用单端通道，将绕过增益放大器。

通过设置 ADCSRA 寄存器的 ADEN 即可启动 ADC。只有当 ADEN 置位时参考电压及输入通道选择才生效。ADEN 清零时 ADC 并不耗电，因此建议在进入节能睡眠模式之前关闭 ADC。

ADC 转换结果为 10 位，存放于 ADC 数据寄存器 ADCH 及 ADCL 中。默认情况下转换结果为右对齐，但可通过设置 ADMUX 寄存器的 ADLAR 变为左对齐。

如果要求转换结果左对齐，且最高只需 8 位的转换精度，那么只要读取 ADCH 就足够了。否则要先读 ADCL，再读 ADCH，以保证数据寄存器中的内容是同一次转换的结果。一旦读出 ADCL， ADC 对数据寄存器的寻址就被阻止了。也就是说，读取 ADCL 之后，即使在读 ADCH 之前又有一次 ADC 转换结束，数据寄存器的数据也不会更新，从而保证了转换结果不丢失。 ADCH 被读出后， ADC 即可再次访问 ADCH 及 ADCL 寄存器。

ADC 转换结束可以触发中断。即使由于转换发生在读取 ADCH 与 ADCL 之间而造成 ADC 无法访问数据寄存器，并因此丢失了转换数据，中断仍将触发。

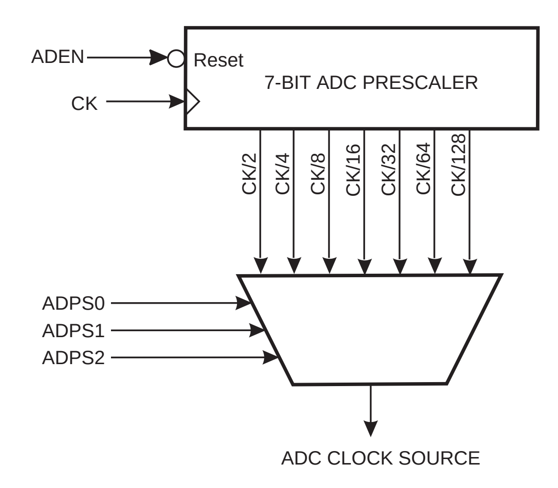
启动一次转换

向 ADC 启动转换位 ADSC 位写 "1" 可以启动单次转换。在转换过程中此位保持为高，直到转换结束，然后被硬件清零。如果在转换过程中选择了另一个通道，那么 ADC 会在改变通道前完成这一次转换。

在连续转换下，持续地进行采样并对 ADC 数据寄存器进行更新。连续转换通过在 ADCSRA 寄存器的 ADFR 位写 1 得到。第一次转换通过向 ADCSRA 寄存器的 ADSC 写 1 来启动。在此模式下，后续的 ADC 转换不依赖于 ADC 中断标志 ADIF 是否置位。

预分频和转换时序

Figure 109. ADC 预分频器



在默认条件下，逐次逼近电路需要一个从 50 kHz 到 200 kHz 的输入时钟以获得最大精度。如果所需的转换精度低于 10 比特，那么输入时钟频率可以高于 200 kHz，以达到更高的

采样率。或者，设置 SFIOR 寄存器的 ADHSM 位允许在更高的功耗下得到更高的 ADC 时钟频率。

ADC 模块包括一个预分频器，它可以由任何超过 100 kHz 的 CPU 时钟来产生可接受的 ADC 时钟。预分频器通过 ADCSRA 寄存器的 ADPS 进行设置。置位 ADCSRA 寄存器的 ADEN 将使能 ADC，预分频器开始计数。只要 ADEN 为 1，预分频器就持续计数，直到 ADEN 清零。

ADCSRA 寄存器的 ADSC 置位后，单端转换在下一个 ADC 时钟周期的上升沿开始启动。有关差分转换时序内容，请见 P 217“ 差分增益通道 ” 。

正常转换需要 13 个 ADC 时钟周期。为了初始化模拟电路， ADC 使能 (ADCSRA 寄存器的 ADEN 置位) 后的第一次转换需要 25 个 ADC 时钟周期。

在普通的 ADC 转换过程中，采样保持在转换启动之后的 1.5 个 ADC 时钟开始；而第一次 ADC 转换的采样保持则发生在转换启动之后的 13.5 个 ADC 时钟。转换结束后，ADC 结果被送入 ADC 数据寄存器，且 ADIF 标志置位。ADSC 同时清零 (单次转换模式)。之后软件可以再次置位 ADSC 标志，从而在 ADC 的第一个上升沿启动一次新的转换。

在连续转换模式下，当 ADSC 为 1 时，只要转换一结束，下一次转换马上开始。转换时间请见 Table 95。

Figure 110. ADC 时序图，第一次转换 (单次转换模式)



Figure 111. ADC 时序图，单次转换

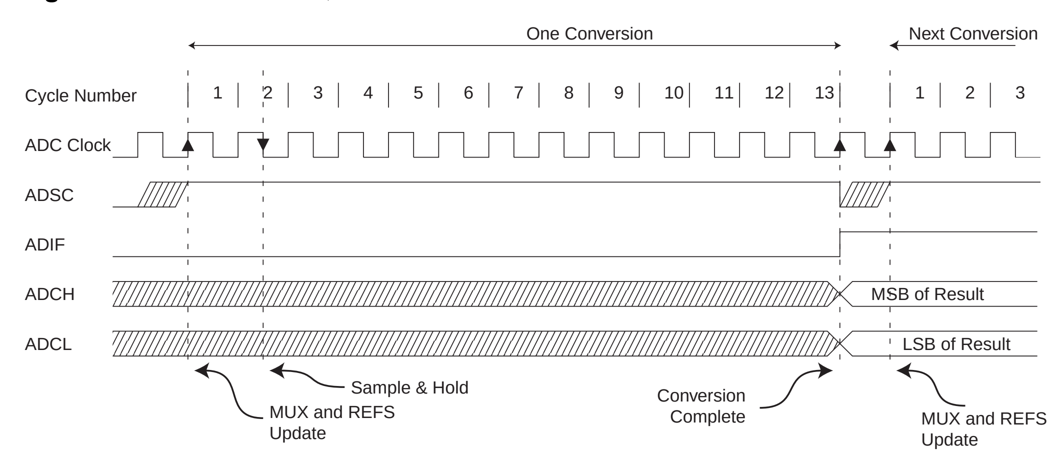




Figure 112. ADC 时序图，连续转换

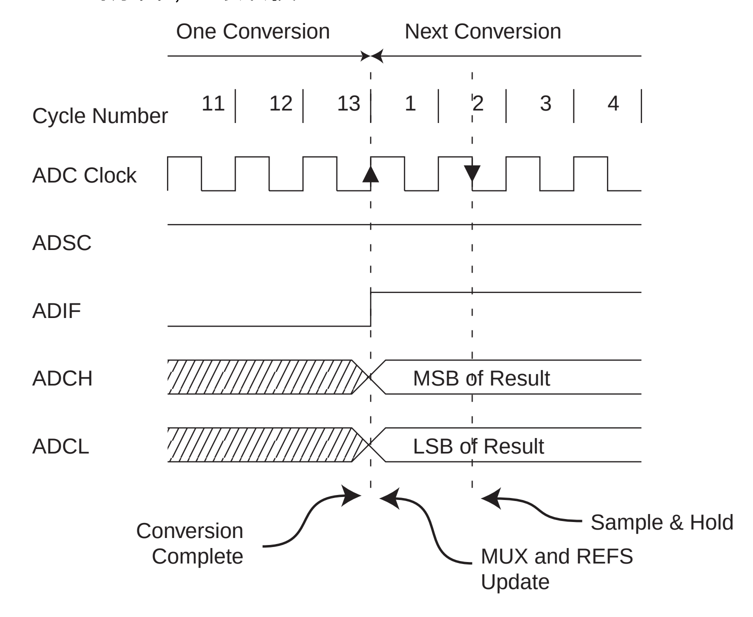


Table 95. ADC 转换时间

条件	采样 & 保持 (启动转换后的时钟周期数)	转换时间 (周期)
第一次转换	14.5	25
正常转换，单端	1.5	13
正常转换，差分	1.5/2.5	13/14

差分增益通道

当使用差分增益通道，需要考虑转换的确定特征。

差分转换与内部时钟 CK_{ADC2} 同步等于 ADC 时钟的一半。同步是当 ADC 接口在 CK_{ADC2} 边沿出现采样与保持时自动实现的。当 CK_{ADC2} 为低时，通过用户启动转换 (即，所有的单次转换与第一次连续转换) 将与单端转换使用相同的时间 (接着的预分频后的 13 个 ADC 时钟周期)。当 CK_{ADC2} 为高时，由于同步机制，将会使用 14 个 ADC 时钟周期。在连续转换模式时，一次转换结束后立即启动新的转换，而由于 CK_{ADC2} 此时为高，所有的自动启动 (即除第一次外) 将使用 14 个 ADC 时钟周期。

在所有的增益设置中，当带宽为 4 kHz 时增益级最优。更高的频率可能会造成非线性放大。当输入信号包含高于增益级带宽的频率时，应在输入前加入低通滤波器。注意，ADC 时钟频率不受增益级带宽限制。比如，不管通道带宽是多少， ADC 时钟周期为 6 µs，允许通道采样率为 12 kSPS。

改变通道或基准源

ADMUX 寄存器中的 MUXn 及 REFS1:0 通过临时寄存器实现了单缓冲。CPU 可对此临时寄存器进行随机访问。这保证了在转换过程中通道和基准源的切换发生于安全的时刻。在转换启动之前通道及基准源的选择可随时进行。一旦转换开始就不允许再选择通道和基准源了，从而保证 ADC 有充足的采样时间。在转换完成 (ADCSRA 寄存器的 ADIF 置位) 之前的最后一个时钟周期，通道和基准源的选择又可以重新开始。转换的开始时刻为 ADSC 置位后的下一个时钟的上升沿。因此，建议用户在置位 ADSC 之后的一个 ADC 时钟周期里，不要操作 ADMUX 以选择新的通道及基准源。

当改变差分通道时要特别注意。一旦选定差分通道，增益级要用 125 µs 来稳定该值。因此在选定新通道后的 125 µs 内不应启动转换。或舍弃该时间段内的转换结果。

当改变 ADC 参考值后 (通过改变 ADMUX 寄存器中的 REFS1:0 位) 的第一次转换也要遵守前面的说明。

若 JTAG 接口使能， PORTF7:4 ADC 通道功能被覆盖，参见 P 77Table 42, " 端口 F 的第二功能 ," 。

ADC 输入通道

选择模拟通道时请注意以下事项：

工作于单次转换模式时，总是在启动转换之前选定通道。在 ADSC 置位后的一个 ADC 时钟周期就可以选择新的模拟输入通道了。但是最简单的办法是等待转换结束后再改变通道。

在连续转换模式下，总是在第一次转换开始之前选定通道。在 ADSC 置位后的一个 ADC 时钟周期就可以选择新的模拟输入通道了。但是最简单的办法是等待转换结束后再改变通道。然而，此时新一次转换已经自动开始了，下一次的转换结果反映的是以前选定的模拟输入通道。以后的转换才是针对新通道的。

当切换到差分增益通道，由于自动偏移抵消电路需要沉积时间，第一次转换结果准确率很低。用户最好舍弃第一次转换结果。

ADC 基准电压源

ADC的参考电压源(V_{REF})反映了ADC的转换范围。若单端通道电平超过了V_{REF}，其结果将接近 0x3FF。 V_{REF} 可以是 AVCC、内部 2.56V 基准或外接于 AREF 引脚的电压。

AVCC通过一个无源开关与ADC相连。片内的2.56V参考电压由能隙基准源(V_{BG})通过内部放大器产生。无论是哪种情况， AREF 都直接与 ADC 相连，通过在 AREF 与地之间外加电容可以提高参考电压的抗噪性。 V_{REF} 可通过高输入内阻的伏特表在 AREF 引脚测得。由于 V_{REF} 的阻抗很高，因此只能连接容性负载。

如果将一个固定电源接到 AREF 引脚，那么用户就不能选择其他的基准源了，因为这会导致片内基准源与外部参考源的短路。如果 AREF 引脚没有联接任何外部参考源，用户可以选择 AVCC 或 1.1V 作为基准源。参考源改变后的第一次 ADC 转换结果可能不准确，建议用户不要使用这一次的转换结果。

如果使用差分通道，选择参考电压不应接近如 P 306Table 136 中所示的 AVCC。

ADC 噪声抑制器

ADC的噪声抑制器使其可以在睡眠模式下进行转换，从而降低由于CPU及外围I/O设备噪声引入的影响。噪声抑制器可在 ADC 降噪模式及空闲模式下使用。为了使用这一特性，应采用如下步骤：

1. 确定 ADC 已经使能，且没有处于转换状态。工作模式应该为单次转换，并且 ADC 转换结束中断使能。
2. 进入 ADC 降噪模式 (或空闲模式)。一旦 CPU 被挂起， ADC 便开始转换。
3. 如果在ADC转换结束之前没有其他中断产生，那么ADC中断将唤醒CPU并执行 ADC 转换结束中断服务程序。如果在 ADC 转换结束之前有其他的中断源唤醒了 CPU，对应的中断服务程序得到执行。 ADC 转换结束后产生 ADC 转换结束中断请求。 CPU 将工作到新的休眠指令得到执行。

进入除空闲模式及 ADC 降噪模式之外的其他休眠模式时， ADC 不会自动关闭。在进入这些休眠模式时，建议将 ADEN 清零以降低功耗。如果 ADC 在该休眠模式下使能，且用户要完成差分转换，建议关闭 ADC 且在唤醒后促使外部转换得到有效值。

模拟输入电路

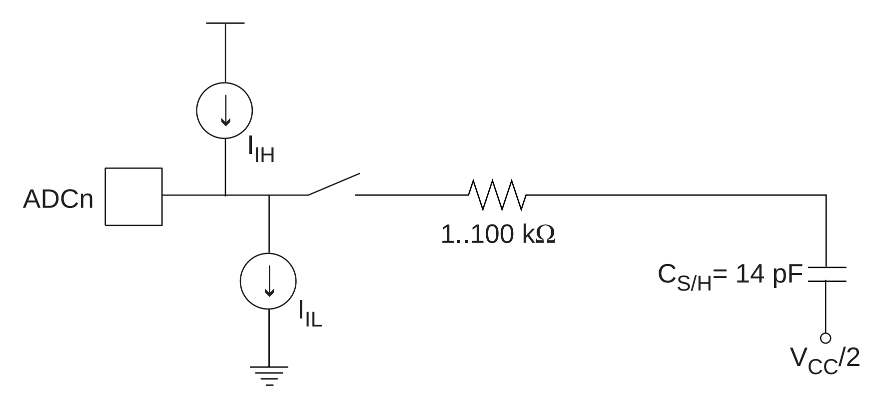
单端通道的模拟输入电路见 Figure 113. 。不论是否用作 ADC 的输入通道，输入到 ADCn 的模拟信号都受到引脚电容及输入泄露的影响。用作 ADC 的输入通道时，模拟信号源必须通过一个串联电阻 (输入通道的组合电阻) 驱动采样保持 (S/H) 电容。

ADC针对那些输出阻抗接近于10 kΩ或更小的模拟信号做了优化。对于这样的信号采样时间可以忽略不计。若信号具有更高的阻抗，那么采样时间就取决于对 S/H 电容充电的时间。这个时间可能变化很大。建议用户使用输出阻抗低且变化缓慢的模拟信号，因为这可以减少对 S/H 电容的电荷传输。

如果使用差分增益通道，输入电路有所不同，建议使用几百 kΩ 的源电阻。

频率高于奈奎斯特频率 ($f_{ADC}/2$) 的信号源不能用于任何一个通道，这样可以避免不可预知的信号卷积造成的失真。在把信号输入到 ADC 之前最好使用一个低通滤波器来滤掉高频信号。

Figure 113. 模拟输入电路

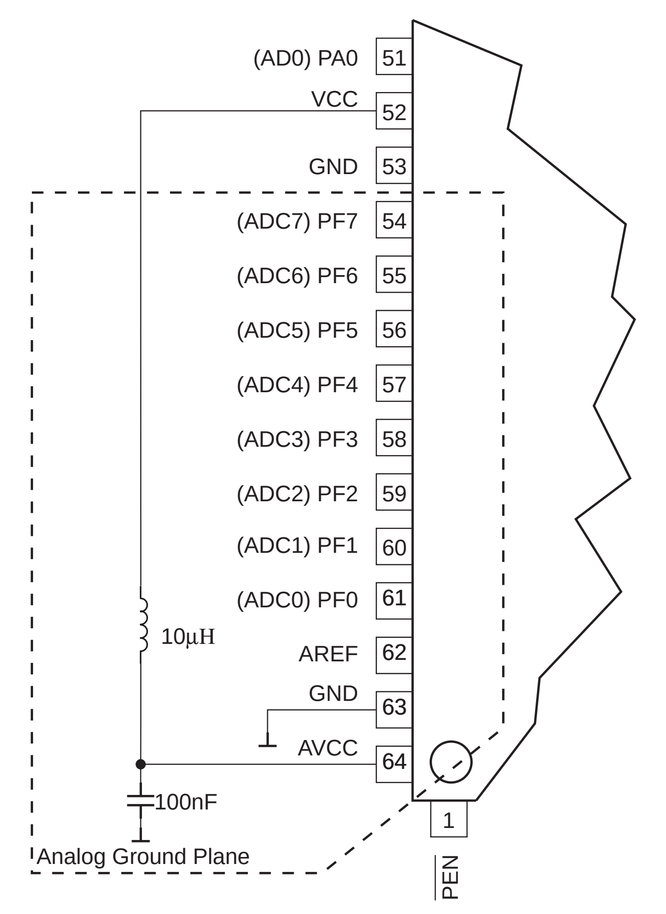


模拟噪声抑制技术

设备内部及外部的数字电路都会产生电磁干扰 (EMI)，从而影响模拟测量的精度。如果转换精度要求较高，那么可以通过以下方法来减少噪声：

1. 模拟通路越短越好。保证模拟信号线位于模拟地之上，并使它们与高速切换的数字信号线分开。
2. 如 Figure 114 所示， AVCC 应通过一个 LC 网络与数字电压源 V_{CC} 连接。
3. 使用 ADC 噪声抑制器来降低来自 CPU 的干扰噪声。
4. 如果有 ADC[3..0] 端口被用作数字输出，那么必须保证在转换进行过程中它们不会有电平的切换。

Figure 114. ADC 电源连接图



偏置补偿方案

增益级有固定的偏置补偿电路，尽可能降低差分度量偏差。模拟电路中的残余偏差可直接由选择的均为差分输入的相同的通道测得。该值可由软件从测量结果中减去。使用这类基于偏置修正的软件，通道的偏置可降到 1 LSB 以下。

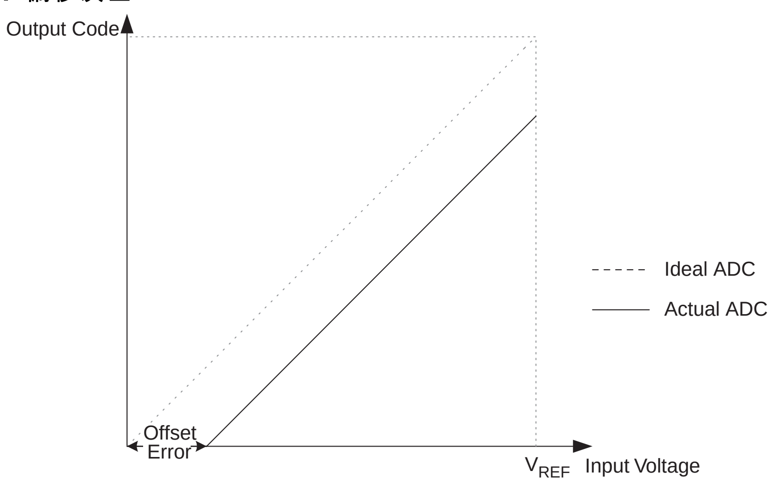
ADC 精度定义

一个 n 位的单端 ADC 将 GND 与 V_{REF} 之间的线性电压转换成 2^n 个 (LSBs) 不同的数字量。最小的转换码为 0，最大的转换码为 2^n-1。

以下几个参数描述了与理想情况之间的偏差：

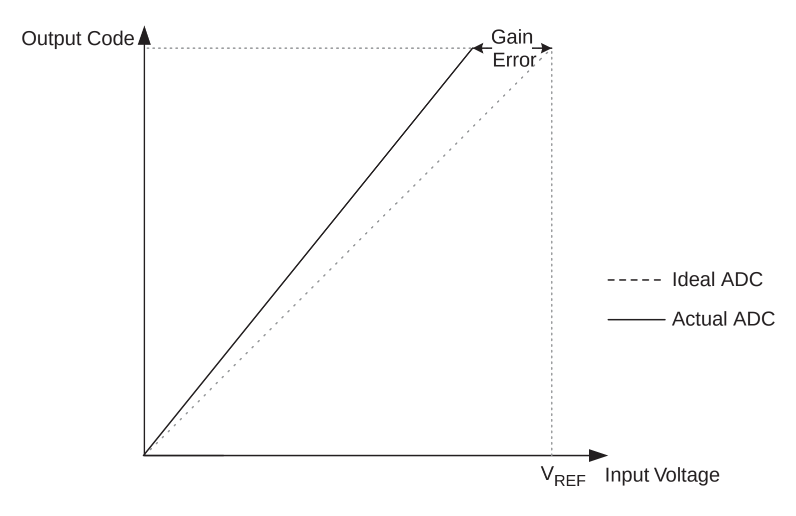
- 偏移 : 第一次转换 (0x000 到 0x001) 与理想转换 (0.5 LSB) 之间的偏差。理想情况 : 0 LSB。

Figure 115. 偏移误差



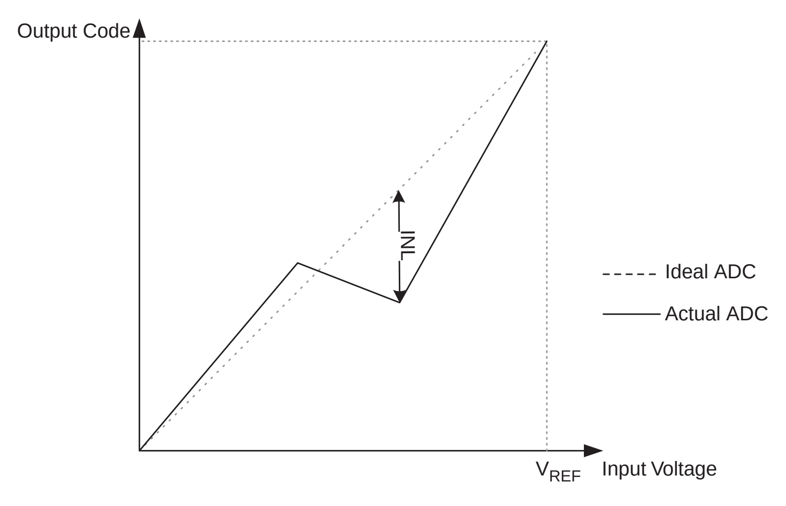
- 增益误差 : 调整偏差之后，最后一次转换 (0x3FE 到 0x3FF) 与理想情况 (最大值以下 1.5 LSB) 之间的偏差即为增益误差。理想值为 0 LSB。

Figure 116. 增益误差



- 整体非线性 (INL)：调整偏移及增益误差之后，所有实际转换与理想转换之间的最大误差即为 INL。理想值：0 LSB。

Figure 117. 整体非线性 (INL)

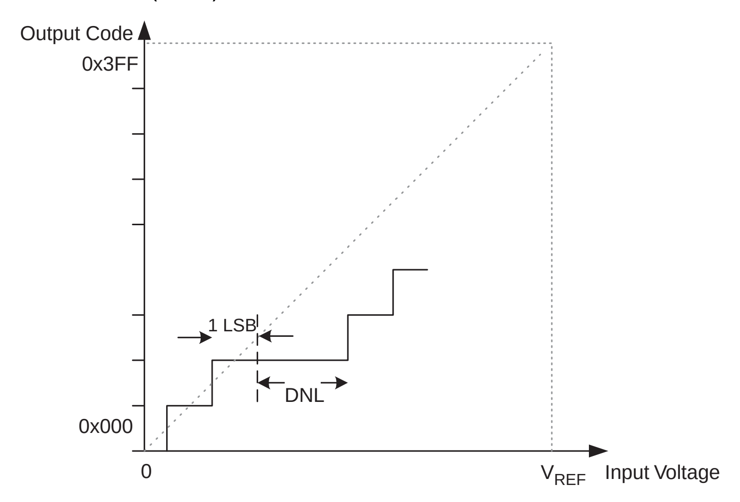






- 差分非线性(DNL):实际码宽(两个邻近转换之间的码间距)与理论码宽(1 LSB)之间的偏差。理论值：0 LSB。

Figure 118. 差分非线性 (DNL)



- 量化误差：由于输入电压被量化成有限位的数码，某个范围的输入电压 (1 LSB) 被转换为相同的数码。量化误差总是为 ± 0.5 LSB。
- 绝对精度：所有实际转换 (未经调整) 与理论转换之间的最大偏差。由偏移、增益误差、差分误差、非线性及量化误差构成。理想值为 ± 0.5 LSB。

ADC 转换结果

转换结束后 (ADIF 为高)，转换结果被存入 ADC 结果寄存器 (ADCL, ADCH)。

单次转换的结果如下：

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

式中 ,V_{IN} 为被选中引脚的输入电压 ,V_{REF} 为参考电压 (参见 P 226Table 97 与 P 227Table 98)。0x000 代表模拟地电平， 0x3FF 代表所选参考电压的数值减去 1LSB。

如果使用差分通道，结果是：

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot GAIN \cdot 512}{V_{REF}}$$

式中，V_{POS} 为输入引脚正电压，V_{NEG} 为输入引脚负电压，GAIN 为选定的增益因子，且 V_{REF} 为参考电压。结果用 2 的补码形式表示 ,从 0x200 (-512d) 到 0x1FF (+511d)。如果用户希望对结果执行快速极性检测，它充分读结果的 MSB(ADCH 中 ADC9)。如果该位为 1，结果为负；否则结果为正。Figure 119 给出差分输入域的解码。

Table 96 给出当选定的增益为 GAIN 且参考电压为 V_{REF} 的差分输入对 (ADCn - ADCm) 的输出码的结果。

Figure 119. 差分测量范围

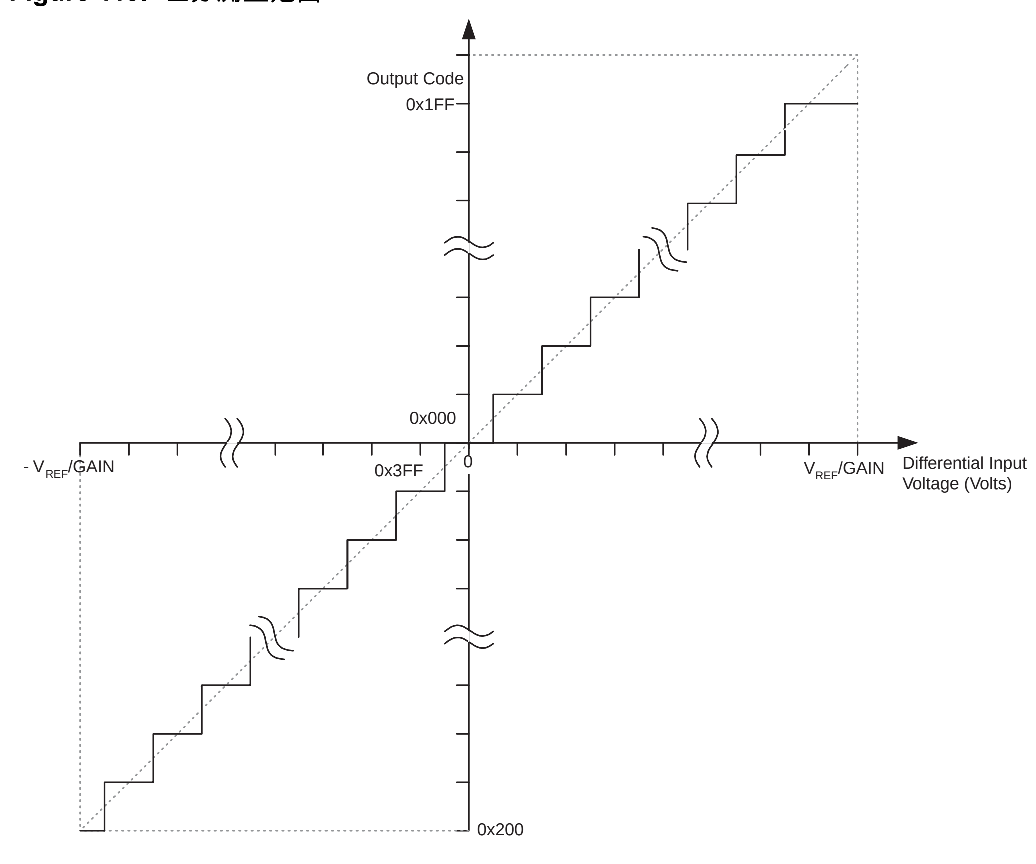




Table 96. 输入电压域输出码的相互关系

V_{ADCn}	读出码	相应十进制值
V_{ADCm} + V_{REF} /GAIN	0x1FF	511
V_{ADCm} + 511/512 V_{REF} /GAIN	0x1FF	511
V_{ADCm} + 511/512 V_{REF} /GAIN	0x1FE	510
...	...	...
V_{ADCm} + 1/512 V_{REF} /GAIN	0x001	1
V_{ADCm}	0x000	0
V_{ADCm} - 1/512 V_{REF} /GAIN	0x3FF	-1
...	...	...
V_{ADCm} - 511/512 V_{REF} /GAIN	0x201	-511
V_{ADCm} - V_{REF} /GAIN	0x200	-512

例：

ADMUX = 0xED (ADC3 - ADC2， 10x 增益， 2.56V 参考电压，左对齐)。

ADC3 上电压为 300 mV， ADC2 电压为 500 mV。

ADCR = 512 * 10 * (300 - 500) / 2560 = -400 = 0x270

ADCL 将读为 0x00，且 ADCH 读为 0x9C。给 ADLAR 写 0 右对齐：ADCL = 0x70，ADCH = 0x02。

ADC 多工选择寄存器 - ADMUX

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7:6 – REFS1:0: 参考电压选择**

如 Table 97 所示，通过这几位可以选择参考电压。如果在转换过程中改变了它们的设置，只有等到当前转换结束 (ADCSRA 寄存器的 ADIF 置位) 之后改变才会起作用。如果在 AREF 引脚上施加了外部参考电压，内部参考电压就不能被选用了。

Table 97. ADC 参考电压选择

REFS1	REFS0	参考电压选择
0	0	AREF，内部 Vref 关闭
0	1	AVCC， AREF 引脚外加滤波电容
1	0	保留
1	1	2.56V 的片内基准电压源， AREF 引脚外加滤波电容

- **Bit 5 – ADLAR: ADC 转换结果左对齐**

ADLAR影响ADC转换结果在ADC数据寄存器中的存放形式。ADLAR置位时转换结果为左对齐，否则为右对齐。 ADLAR 的改变将立即影响 ADC 数据寄存器的内容，不论是否有转换正在进行。关于这一位的完整描述请见 P 228“ADC 数据寄存器 - ADCL 和 ADCH” 。

- **Bits 4:0 – MUX4:0: 模拟通道与增益选择位**

通过这几位的设置，可以对连接到 ADC 的模拟输入进行选择。也可对差分通道增益进行选择。细节见 Table 98。如果在转换过程中改变这几位的值，那么只有到转换结束 (ADCSRA 寄存器的 ADIF 置位) 后新的设置才有效。

Table 98. 输入通道与增益选择

MUX4..0	单端输入	正差分输入	负差分输入	增益
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000		ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010	N/A	ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.23V (V_{BG})	N/A		
11111	0V (GND)			

ADC 控制和状态寄存器 A - ADCSRA

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – ADEN: ADC 使能**

ADEN置位即启动ADC ,否则ADC功能关闭。在转换过程中关闭ADC将立即中止正在进行的转换。

- **Bit 6 – ADSC: ADC 开始转换**

在单次转换模式下，ADSC 置位将启动一次 ADC 转换。在连续转换模式下，ADSC 置位将启动首次转换。第一次转换 (在 ADC 启动之后置位 ADSC , 或者在使能 ADC 的同时置位 ADSC) 需要 25 个 ADC 时钟周期，而不是正常情况下的 13 个。第一次转换执行 ADC 初始化的工作。

在转换进行过程中读取 ADSC 的返回值为 "1" , 直到转换结束。ADSC 清零不产生任何动作。

- **Bit 5 – ADFR: ADC 连续转换选择**

当该位写 1， ADC 工作在连续转换模式。在该模式下， ADC 不断对 数据寄存器采样与更新。该位写 0，停止连续转换模式。

- **Bit 4 – ADIF: ADC 中断标志**

在 ADC 转换结束，且数据寄存器被更新后， ADIF 置位。如果 ADIE 及 SREG 中的全局中断使能位 I 也置位，ADC 转换结束中断服务程序即得以执行，同时 ADIF 硬件清零。此外，还可以通过向此标志写 1 来清 ADIF。要注意的是，如果对 ADCSRA 进行读 - 修改 - 写操作，那么待处理的中断会被禁止。这也适用于 SBI 及 CBI 指令。

- **Bit 3 – ADIE: ADC 中断使能**

若 ADIE 及 SREG 的位 I 置位， ADC 转换结束中断即被激活。

- **Bits 2:0 – ADPS2:0: ADC 预分频器选择位**

这几位确定了 XTAL 与 ADC 输入时钟之间的分频因子。

Table 99. ADC 预分频选择

ADPS2	ADPS1	ADPS0	分频因子
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ADC 数据寄存器 - ADCL 和 ADCH

ADLAR = 0:

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
读 / 写	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1:

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
读 / 写	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
初始值	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADC 转换结束后，转换结果存于这两个寄存器之中。如果采用差分通道，结果由 2 的补码形式表示。

读取 ADCL 之后，ADC 数据寄存器一直要等到 ADCH 也被读出才可以进行数据更新。因此，如果转换结果为左对齐，且要求的精度不高于 8 比特，那么仅需读取 ADCH 就足够了。否则必须先读出 ADCL 再读 ADCH。

ADMUX 寄存器的 ADLAR 及 MUXn 会影响转换结果在数据寄存器中的表示方式。如果 ADLAR 为 1，那么结果为左对齐；反之 (系统缺省设置)，结果为右对齐。

- **ADC9:0: ADC 转换结果**

ADC 转换的结果，细节见 P 224“ADC 转换结果 ”。



JTAG 接口和片上调试系统 OCD(On-chip Debug)

特点

- **与 IEEE 1149.1 标准兼容的 JTAG 接口**
- **遵从 IEEE 1149.1 (JTAG) 标准的边界扫描功能**
- **调试器可以访问：**
  - **所有的片内外设**
  - **内部和外部 SRAM**
  - **寄存器文件**
  - **程序计数器**
  - **EEPROM 和 Flash 存储器**
- **扩展 OCD 支持各种断点，包括**
  - **AVR *Break* 指令**
  - **程序流程改变**
  - **单步**
  - **单个地址或一个地址范围的程序断点**
  - **单个地址或一个地址范围的数据断点**
- **通过 JTAG 接口对 Flash、 EEPROM、熔丝位和锁定位进行编程**
- **AVR Studio 支持 OCD**

综述

与 IEEE 1149.1 标准兼容的 AVR JTAG 接口可用于：

- 通过 JTAG 边界扫描功能测试 PCB。
- 对非易失性存储器、熔丝位和锁定位进行编程。
- 片上调试 OCD。

下面为 JTAG 接口的简短描述。有关通过 JTAG 接口进行编程、使用边界扫描链的具体说明请分别参见 P 286“ 通过 JTAG 接口进行编程 ” 及 P 235“IEEE 1149.1 (JTAG) 边界扫描 ” 。片内调试功能 OCD 以专用的 JTAG 指令实现 ,只在 ATMEL 内部分发。ATMEL 将有选择地支持第三方 JTAGICE 生产商。

Figure 120为JTAG接口及OCD系统的框图。TAP控制器为受TCK和TMS信号控制的状态机。 TAP 控制器或者选择 JTAG 指令寄存器，或者选择数据寄存器作为 TDI(输入) 和 TDO(输出) 之间的扫描链 (移位寄存器)。指令寄存器保存了控制数据寄存器行为的 JTAG 指令。

ID 寄存器、旁路 (Bypass) 寄存器和边界扫描链组成了用于板级测试的数据寄存器。JTAG 编程接口 (包括几个物理的和虚拟的数据寄存器) 用于串行编程。片内扫描链和断点扫描链只用于 OCD 功能。

测试访问端口 - TAP

JTAG接口有四个引脚。以JTAG的术语来说 ,这些引脚组成了测试访问端口TAP。这些引脚是：

- TMS：测试模式选择。此引脚用来实现 TAP 控制器各个状态之间的切换。
- TCK：测试时钟。 JTAG 操作是与 TCK 同步的。
- TDI：测试数据输入--需要移位到指令寄存器或数据寄存器(扫描链)的串行输入数据。
- TDO：测试数据输出 -- 自指令寄存器或数据寄存器串行移出的数据。

ATmega128 没有实现 IEEE 1149.1 标准指定的可选 TAP 信号 TRST - Test ReSeT。

JTAGEN 没有编程时，四个 TAP 引脚为普通 I/O 引脚， TAP 控制器处于复位状态。若 JTAGEN 被编程且 MCUCSR 寄存器的 JTD 位清零 ,TAP 信号由片内上拉电阻拉高 ,可以通过 JTAG 接口进行边界扫描和编程。当 JTAG TAP 控制器不移出数据时， TAP 输出引脚 (TDO) 悬空，因此必须接一个上拉电阻或其他硬件以拉高电压。 JTAGEN 在芯片出厂前即已编程。

对于 OCD 系统，调试器还监控 $\overline{\text{RESET}}$ 引脚以检测外部复位源。调试器也可以将 $\overline{\text{RESET}}$ 拉低以复位整个系统。此时需要注意的是复位线必须是开漏驱动的。

Figure 120. 方框图

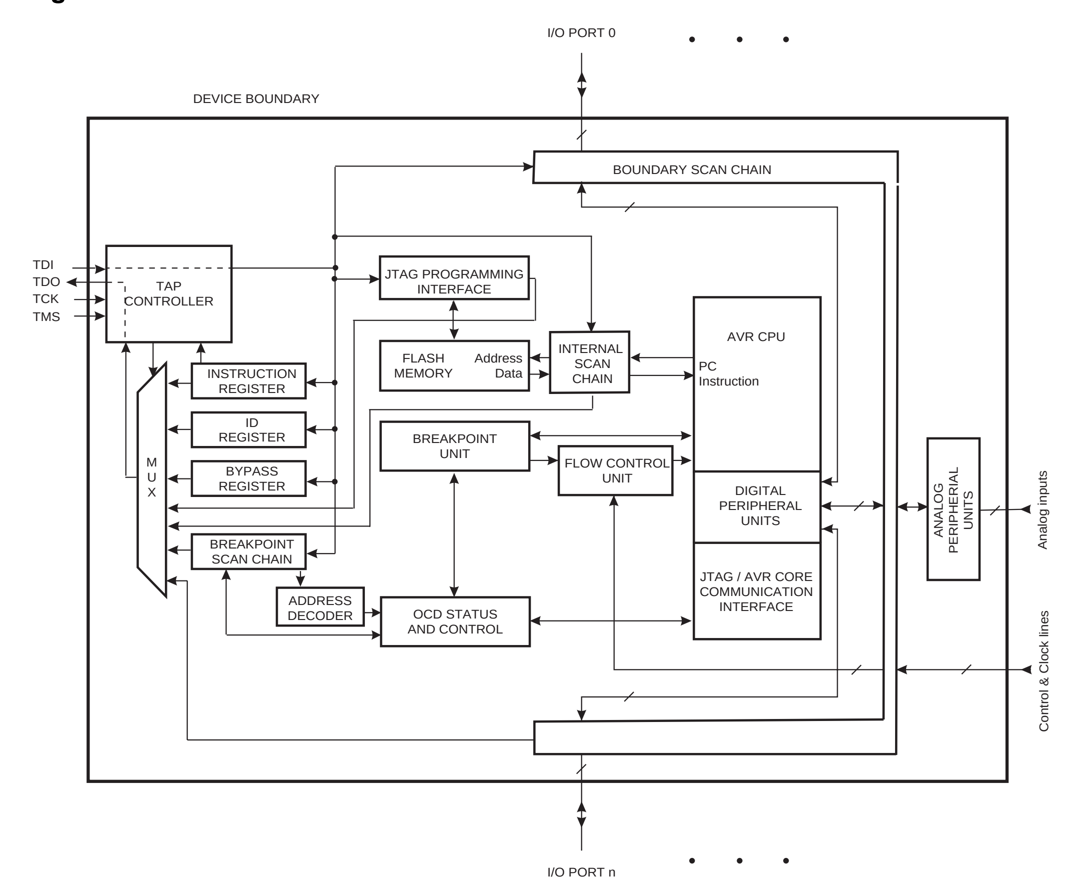


Figure 121. TAP 控制器状态转换框图

TAP 控制器

TAP控制器为具有16个状态的有限状态机，控制边界扫描电路、JTAG编程电路和OCD系统的操作。 Figure 121 给出的状态转换依赖于在 TCK 上升沿输入的 TMS 信号 (临近状态方框的 0/1 数字即是)。上电复位后的初始状态为 Test-Logic-Reset。

按照本手册的定义，移位寄存器的 LSB 首先移入 / 移出。

假定 Run-Test/Idle 为当前状态，使用 JTAG 接口的典型过程为 :

- 在 TCK 的上升沿从 TMS 输入信号序列 1，1，0，0 到移位指令寄存器 – Shift-IR 状态。然后保持 TMS 为低，在 TCK 的上升沿从 TDI 将 4 比特的 JTAG 指令的低 3 位移入 JTAG 指令寄存器。接着将 TMS 拉高，从 TDI 将 JTAG 指令的最高位移入 JTAG指令寄存器。在指令移入的同时捕捉到的IR状态0x01从TDO输出。JTAG指令选择一个特定的数据寄存器作为 TDI 和 TDO 之间的通路，并控制围绕着被选寄存器的电路。

- 在 TMS 上施加信号序列 1，1，0 以重新进入 Run-Test/Idle 状态。指令在 Update-IR 状态从移位寄存器锁存到并行输出。状态 Exit-IR、 Pause-IR、和 Exit2-IR 只用于切换状态。
- 在 TCK 的上升沿从 TMS 输入信号序列 1，0，0 到移位数据寄存器 – Shift-DR 状态。然后在 TCK 的上升沿从 TDI 将数据输入到被选择的数据寄存器 (由位于 JTAG 指令寄存器的 JTAG 指令选定)。为了保持于 Shift-DR 状态，除了输入数据的最高位，TMS 一直要保持为低电平。同时，在 Capture-DR 状态下捕捉到的数据寄存器的并行输入自 TDO 输出。
- 在 TCK 的上升沿从 TMS 输入信号序列 1，0，0 到移位数据寄存器 – Shift-DR 状态。然后在 TCK 的上升沿从 TDI 将数据输入到被选择的数据寄存器 (由位于 JTAG 指令寄存器的 JTAG 指令选定)。为了保持于 Shift-DR 状态，除了输入数据的最高位，TMS 一直要保持为低电平。同时，在 Capture-DR 状态下捕捉到的数据寄存器的并行输入自 TDO 输出。
- 在 TMS 上施加信号序列 1，1，0 以重新进入 Run-Test/Idle 状态。如果选定的数据寄存器有锁存的并行输出，锁存操作在 Update-DR 状态完成。状态 Exit-IR、 Pause-IR、和 Exit2-IR 只用于切换状态。

如状态转换图所示，在选择 JTAG 指令和使用数据寄存器之间不需要进入 Run-Test/Idle 状态。一些 JTAG 指令可能在 Run-Test/Idle 状态执行某些工作，使之不再适合称作 Idle 状态。

Note: 与 TAP 控制器初始状态不同，Test-Logic-Reset 状态总是可以进入的，只要将 TMS 保持为高 5 个 TCK 时钟就可以。

有关 JTAG 规范的具体信息请参考 P 234“ 参考书目 ” 。

使用边界扫描链

完整的边界扫描说明在 P 235“IEEE 1149.1 (JTAG) 边界扫描 ” 。

利用片上调试系统 OCD

如 Figure 120 所示，支持 OCD 功能的硬件主要包括：
- 界于 AVR CPU 和外设单元之间的接口的扫描链。
- 断点单元。
- CPU 和 JTAG 系统之间的通讯接口。

所有实现调试功能所需的读或修改 / 写操作都通过片内 AVR CPU 扫描链施加 AVR 指令的方式实现。然后 CPU 将结果发送到一个特定的 I/O 存储器地址，此 I/O 存储器地址为 CPU 和 JTAG 通讯接口的一部分。

断点单元实现了如下断点功能：程序跳转、单步、两个程序存储器断点以及两个组合断点。 4 个断点可以实现如下配置：
- 4 个程序存储器断点
- 3 个程序存储器断点 + 1 个数据存储器断点
- 2 个程序存储器断点 + 2 个数据存储器断点
- 2 个程序存储器断点 + 1 个屏蔽 (范围) 程序存储器断点
- 2 个程序存储器断点 + 1 个屏蔽 (范围) 数据存储器断点

AVR Studio® 这样的调试器可能使用了其中的一些资源，减少了最终用户的灵活性。

与 OCD 相关的指令列于 P 233“OCD 指定的 JTAG 指令 ” 。

为了使能 JTAG 测试访问端口， JTAG 使能位 JTAGEN 必须置位，而且不能设置任何一个锁定位以保证 OCD 的工作。这种一旦有锁定位被编程即禁止 OCD 功能的方式实现了程序保密的要求。否则 OCD 系统就为加密器件提供了一个后门。

AVR Studio可以使用户完全控制AVR程序的执行，不管是利用JTAG的OCD功能，还是利用 AVR ICE，或是 AVR 指令集仿真器。AVR Studio 支持源代码级的仿真，包括用 Atmel 提供的 AVR 汇编器写的汇编程序，以及用第三方的 C 编译器写的 C 程序。

AVR Studio 可以运行于 Microsoft Windows® 95/98/2000 以及 Microsoft WindowsNT®。

AVR Studio 的完整说明请参考 **AVR Studio User Guide**。

AVR Studio包含了所有必须的运行命令，不论是源代码级还是汇编级。用户可以运行诸如单步、跟踪进入、运行到光标处、停止、复位等各种与其他调试器完全相同的操作。此外，用户还可以通过 BREAK 指令设置无限多个程序断点，以及设置两个数据存储器断点，或组成一个屏蔽(范围)断点。

OCD 指定的 JTAG 指令

OCD 功能是以专有 JTAG 指令实现的。

PRIVATE0; $8

专有的 JTAG 指令以访问 OCD 系统。

PRIVATE1; $9

专有的 JTAG 指令以访问 OCD 系统。

PRIVATE2; $A

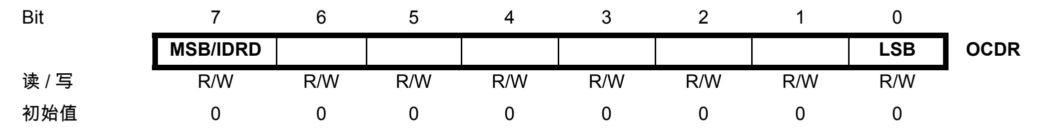
专有的 JTAG 指令以访问 OCD 系统。

PRIVATE3; $B

专有的 JTAG 指令以访问 OCD 系统。

I/O 存储器中与 OCD 相关的寄存器

OCD 寄存器 - OCDR



Bit	7	6	5	4	3	2	1	0	
	MSB/IDRD							LSB	OCDR
读 / 写	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

寄存器 OCDR 为正在处理器内运行的程序与调试器提供了一个通讯通道。CPU 可以在此位置写入需要传递给调试器的数据。同时内部标志 – I/O 调试寄存器被修改标志 – IDRD 置位，告诉调试器这个寄存器已经有了新数据。当 CPU 读 OCDR 寄存器时，低 7 位来自寄存器 OCDR，而 MSB 为 IDRD。调试器读取数据后 IDRD 即清零。

一些 AVR 器件将此寄存器与其他标准 I/O 位置共享。此时，只有在 OCDEN 已经编程和调试器使能访问的时候才可以访问寄存器 OCDR。其他时候访问的是标准 I/O。

请参考调试器文档以获取使用这个寄存器的进一步的信息。

利用 JTAG 的编程能力

通过 JTAG 对 AVR 器件进行编程是由 JTAG 端口的 TCK、 TMS、 TDI 和 TDO 实现的。这就是除了电源以外进行 JTAG 编程所需要的所有引脚。同时，熔丝位 JTAGEN 必须已被编程，而且寄存器 MCUSR 的 JTD 必须清零，以使能 JTAG 测试访问端口。

通过 JTAG 可以实现如下的编程功能：

- Flash 编程及校验
- EEPROM 编程及校验
- 熔丝位编程及校验
- 锁定位编程及校验

锁定位的安全性与并行编程模式是完全一样的。如果 LB1 或 LB2 被编程，熔丝位 OCDEN 就不能再编程了，除非首先进行芯片擦除。从而消除了程序安全的后门。

通过 JTAG 进行编程的特定 JTAG 指令列于 P 286“ 通过 JTAG 接口进行编程 ” 。

参考书目

更多的有关边界扫描的信息请参考如下文章：

- IEEE: IEEE Std 1149.1-1990. IEEE Standard Test Access Port and Boundary-scan Architecture, IEEE, 1993
- Colin Maunder: The Board Designers Guide to Testable Logic Circuits, Addison-Wesley, 1992

IEEE 1149.1 (JTAG) 边界扫描

特点

- **与 IEEE 1149.1 标准相兼容的 JTAG 接口**
- **遵从 JTAG 标准的边界扫描功能**
- **可以扫描所有的端口**
- **支持可选的 IDCODE 指令**
- **额外的 AVR_RESET 指令以复位 AVR**

系统综述

通过边界扫描链可以驱动和确定 AVR 数字 I/O 引脚的逻辑电平，以及可以脱离系统停止工作的模拟电路的边界。从系统一级来说，所有具有 JTAG 功能的 IC 都以串行的方式与 TDI/TDO 信号相连，形成一个很长的移位寄存器。外部控制器可以设置器件，使之在输出引脚输出信号，以及确定器件输入引脚的逻辑电平。然后控制器可以比较接收到的数据和期望值。通过这种方式，边界扫描只用 4 个 TAP 信号实现了 PCB 板上器件互连、器件完整性的测试。

四个 IEEE 1149.1 要求实现的 JTAG 指令：IDCODE、 BYPASS、 SAMPLE/PRELOAD 和EXTEST，以及AVR的公共JTAG指令AVR_RESET 可以用来测试PCB。由于IDCODE 是缺省的 JTAG 指令，因此对数据寄存器通路的初始扫描将得到器件的 ID 号。在测试过程当中保持 AVR 为复位状态是一个不错的选择。如果不处于复位状态，器件的输入将由扫描操作决定，而且退出测试模式时芯片内的软件有可能处于不确定的状态。处于复位状态时，端口为高阻态。在需要的时候可以通过 BYPASS 指令来缩短扫描链。使 AVR 进入复位状态有两种方法，一是将 AVR $\overline{\text{RESET}}$ 引脚拉低，二是给 AVR 复位寄存器加载合适的数值并执行 AVR_RESET 指令。

EXTEST 指令用来读取引脚上的数据，以及将数据加载到输出引脚上。一旦 EXTEST 指令加载到 JTAG IR 寄存器，输出锁存的数据就从引脚输出。因此，为了防止在第一次执行 EXTEST 指令时损坏电路板，需要使用 SAMPLE/PRELOAD 指令来为整个扫描环路设置初始数据。 SAMPLE/PRELOAD 还可以用来给正常工作的 AVR 拍一个外部引脚的“ 快照”。

为了使能JTAG测试访问端口TAP，JTAG使能位JTAGEN必须置位，而寄存器MCUCSR 的 JTD 必须清零。

在使用 JTAG 接口进行边界扫描时， JTAG TCK 时钟频率可以比芯片的工作频率高。此时芯片的时钟无需运行。

数据寄存器

与边界扫描操作相关的数据寄存器有：

- Bypass 寄存器
- 器件识别寄存器
- 复位寄存器
- 边界扫描链

旁路寄存器

Bypass寄存器由一个移位寄存器构成。当Bypass寄存器被选中为TDI和TDO之间的通路时，退出 Capture-DR 状态将使寄存器复位为 0。测试其他器件时 Bypass 寄存器可以用来缩短系统的扫描链。

器件识别寄存器

Figure 122 为器件识别寄存器的结构。

Figure 122. 器件识别寄存器的格式

Bit	31 ... 28 (MSB)	27 ... 12	11 ... 1	0 (LSB)
器件 ID	**版本**	**器件型号**	**生产商 ID**	**1**
	4 bits	16 bits	11 bits	1-bit

版本

版本用了 4 比特，用于识别此器件的版本。版本 A 为 0x0，版本 B 为 0x1 等。

器件型号

型号占了 16 比特。ATmega128 的 JTAG 型号列于 Table 100。

Table 100. AVR JTAG 型号

型号	JTAG 型号 (Hex)
ATmega128	0x9702

生产商 ID

生产商 ID 占用 11 位，用来识别生产商。 ATMEL 的 JTAG 生产商 ID 列于 Table 101。

Table 101. 生产商 ID

生产商	JTAG 生产商 ID (Hex)
ATMEL	0x01F

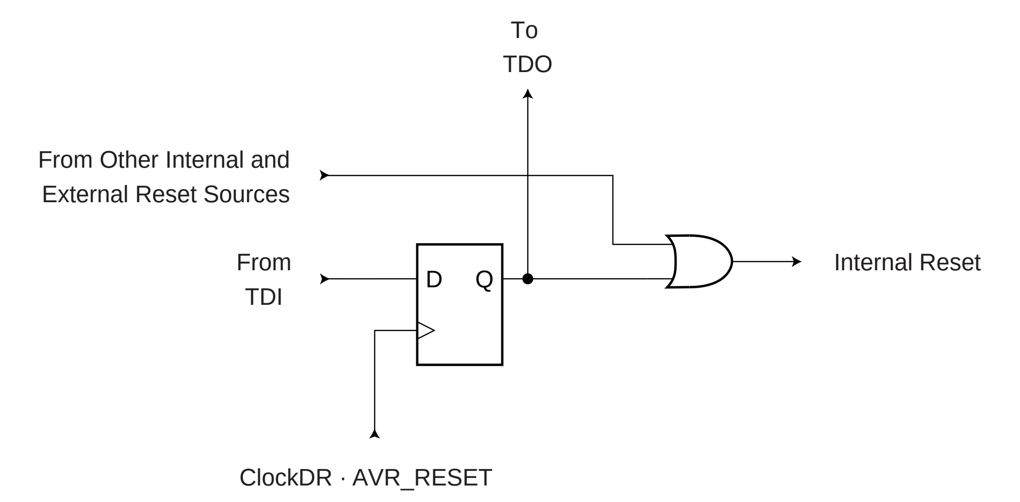
复位寄存器

复位寄存器是用来复位器件的测试数据寄存器。由于复位时 AVR 的端口为高阻态，复位寄存器还可以用来代替没有实现的 JTAG 指令 HIGHZ。

寄存器的数据不为 0 时 AVR 复位引脚将拉低。只要复位寄存器保持这个数值，AVR 将一直保持复位状态。根据时钟设置的不同，在复位 AVR 复位寄存器之后 CPU 还将处于复位状态，直到超出复位延时周期。这个数据寄存器的输出是不锁存的，因此一旦数据移出复位就发生。详见 Figure 123。



Figure 123. 复位寄存器



边界扫描链

边界扫描链可以驱动和确定 AVR 数字 I/O 引脚的逻辑电平，以及可以脱离系统停止工作的模拟电路的边界。

完整说明请见 P 238“ 边界扫描链 ” 一节。

边界扫描指定的 JTAG 指令

指令寄存器的大小为 4 比特 ,支持 16 种功能。下面列出的是支持边界扫描的 JTAG 指令。要注意器件没有实现 HIGHZ 指令。但是可以通过 AVR_RESET 指令将所有具有三态功能的引脚设置为高阻态，因为所有端口的初始状态为三态。

按照本手册的定义，移位寄存器的 LSB 首先移入 / 移出。

每条指令的操作码 (OPCODE) 以 HEX 格式列于指令名称的后面。文字则说明哪一个数据寄存器被选择为 TDI 和 TDO 之间的通路。

EXTEST; $0

这是必须实现的 JTAG 指令，用来选择边界扫描链作为数据寄存器，以测试 AVR 之外的电路。在扫描链里可以访问端口的引脚、上拉控制、输出控制、输出的数据，以及输入的数据。可以脱离芯片停止工作的模拟电路、模拟和数字逻辑之间的接口也包含在扫描链里。一旦 EXTEST 指令加载到 JTAG IR 寄存器，边界扫描链里锁存的输出数据就从输出引脚输出。

有效的状态为 :

- Capture-DR：将引脚上的数据 (电平) 读入边界扫描链。
- Shift-DR：内部扫描链以 TCK 为时钟进行移位操作。
- Update-DR：扫描链上的数据反映到输出引脚上。

IDCODE; $1

这是可选的 JTAG 指令，用来选择 32 位的 ID 寄存器为数据寄存器。 ID 寄存器包括版本号、型号以及由 JEDEC 确定的生产商 ID。这也是上电复位之后的缺省指令。

有效的状态为 :

- Capture-DR：IDCODE 寄存器的数据进入扫描链。
- Shift-DR：IDCODE 扫描链以 TCK 为时钟进行移位。

SAMPLE_PRELOAD; $2

这是必须实现的 JTAG 指令 , 用来预先加载输出锁存器 , 并在不影响系统工作的前提下为输入 / 输出引脚拍一个状态 “ 快照” 。此时锁存的输出并没有连接到输出引脚上。边界扫描链被选择为数据寄存器。

有效的状态为 :

- Capture-DR：将引脚上的数据 (电平) 读入边界扫描链。
- Shift-DR：边界扫描链以 TCK 为时钟进行移位操作。
- Update-DR：扫描链上的数据加载到输出锁存器。但是锁存的输出并不连接到输出引脚上。

AVR_RESET; $C

这是与 AVR 相关的公共 JTAG 指令 ,用来强制 AVR 进入复位模式 ,或者是释放 JTAG 复位源。 TAP 控制器本身不受此指令影响。一比特宽的复位寄存器被选择为数据寄存器。只要复位扫描链为逻辑 1 ，复位就一直有效。这个扫描链的输出是不锁存的。

有效的状态为 :

- Shift-DR：复位寄存器以 TCK 为时钟进行移位操作。

BYPASS; $F

这是必须实现的 JTAG 指令，用来将 Bypass 寄存器选择为数据寄存器。

有效的状态为 :

- Capture-DR：将逻辑 ‘0’ 加载到 Bypass 寄存器。
- Shift-DR：TDI 和 TDO 之间的 Bypass 寄存器单元进行移位操作。

I/O 存储器里与边界扫描相关的寄存器

MCU 控制和状态寄存器 - MCUCSR

MCU 控制和状态寄存器包含了 MCU 通用功能的控制位 , 以及提供是什么复位源引起复位的信息。

Bit	7	6	5	4	3	2	1	0	
	JTD	–	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
读 / 写	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	参见各个位的说明					

- **Bit 7 – JTD: JTAG 接口禁用**

当这一位为 '0’ , 且 JTAG 使能熔丝位 JTAGEN 被编程时 JTAG 接口使能；否则 JTAG 接口功能禁止。为了防止无意改变 JTAG 接口的工作状态 , 需要遵守如下时间限制 : 在四个时钟周期之内对这个位两次写入需要的数据。

若 JTAG 接口未与其他 JTAG 电路连接， JTD 位应置为 ’1’ ，以避免 JTAG 接口 TDO 引脚的静态电流。

- **Bit 4 – JTRF: JTAG 复位标志**

若复位是由 JTAG 复位寄存器不为 '0’ 且执行了的 AVR_RESET 指令引发的， JTRF 置位。通过写入 '0’ 或上电复位的方法可以将其清零。

边界扫描链

通过边界扫描链可以驱动和确定 AVR 数字 I/O 引脚的逻辑电平，以及可以脱离系统停止工作的模拟电路的边界。

扫描数字端口引脚

Figure 124为带上拉的双向端口引脚的边界扫描单元。它包括适用于上拉使能 – PUExn功能的标准边界扫描单元以及将输出控制 – OCxn、输出数据 – ODxn 和输入数据 – IDxn 三个信号组合为两阶段移位寄存器的双向引脚单元。下面的说明没有给出端口和引脚的下标。

本数据手册没有包括边界扫描逻辑。 Figure 125 给出了在 P 61“I/O 端口 ” 一节说明的一个简单的数字端口引脚。Figure 124 给出的详细的边界扫描框图即为 Figure 125 的虚线框部分。



不考虑第二功能时，输入数据 – ID 对应 PINxn 寄存器的值 (但是 ID 没有同步器)，输出数据对应 PORT 寄存器，输出控制对应数据方向 – DD 寄存器，上拉使能 – PUExn – 对应于逻辑表达式 $\overline{PUD} \cdot \overline{DDxn} \cdot PORTxn$。

数字第二功能不与 Figure 125 虚线框连接，使得扫描链可以读到实际的引脚数据。对于模拟功能，外部引脚与模拟电路有直接的连接关系，扫描链可以插入到数字电路和模拟电路之间。

Figure 124. 带上拉电阻双向端口引脚的边界扫描单元 .

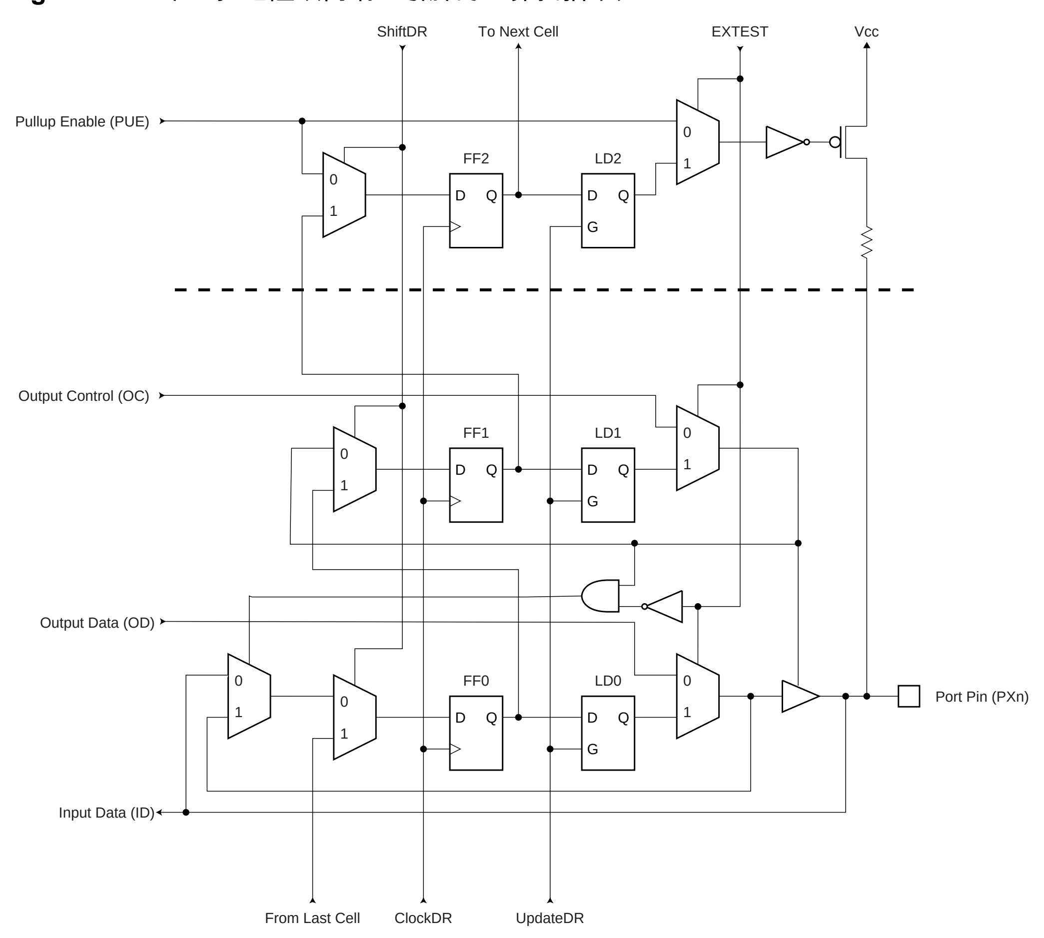


Figure 125. 普通端口引脚的原理图

PUD:	PULLUP DISABLE	WDx:	WRITE DDRx
PUExn:	PULLUP ENABLE for pin Pxn	RDx:	READ DDRx
OCxn:	OUTPUT CONTROL for pin Pxn	WPx:	WRITE PORTx
ODxn:	OUTPUT DATA to pin Pxn	RRx:	READ PORTx REGISTER
IDxn:	INPUT DATA from pin Pxn	RPx:	READ PORTx PIN
SLEEP:	SLEEP CONTROL	$CLK_{I/O}$:	I/O CLOCK

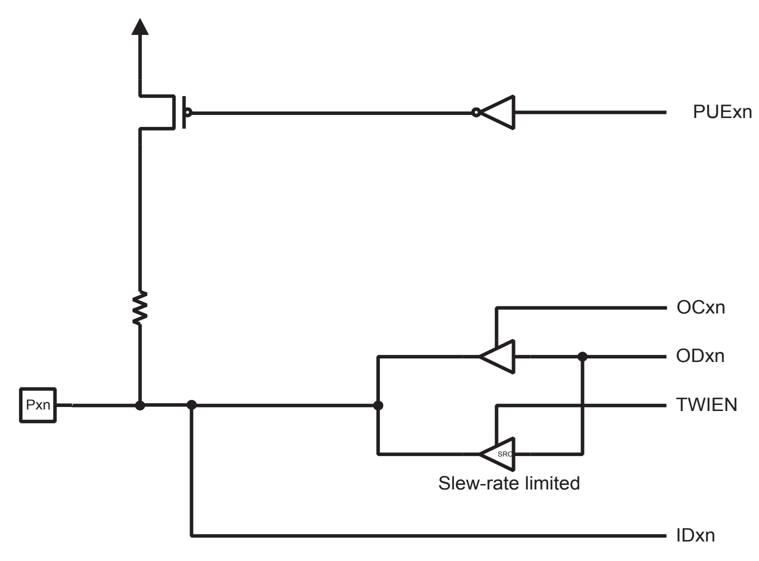
边界扫描和 TWI

两线接口引脚 SCL 和 SDA 在扫描链里有一个额外的控制信号 : 两线接口使能 – TWIEN。如 Figure 126 所示，TWIEN 信号可以使能与普通数字端口并行的、具有斜率控制功能的三态缓冲器，并与 Figure 130 给出的普通扫描链单元相连。

Notes: 1. 器件没有为输入引脚的 50ns 尖峰滤波器提供扫描链。数字端口的扫描支持已经足够用于连接性测试了。在扫描链路里加入 TWIEN 信号的唯一原因是在进行边界扫描时可以关闭斜率控制缓冲器。
2. 不要同时使能 OC 和 TWIEN，否则会引起驱动冲突。



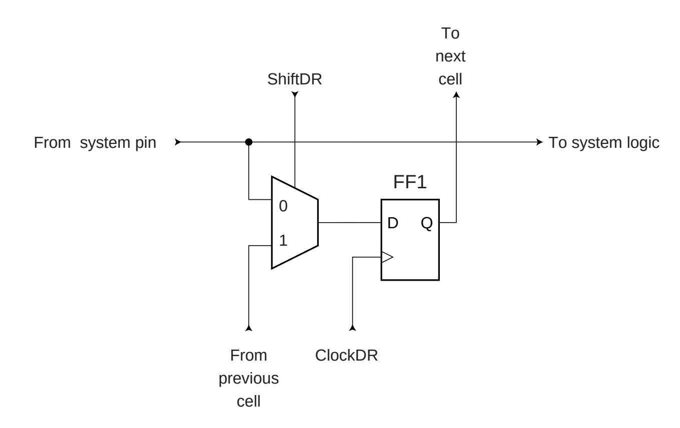
Figure 126. 为两线接口添加的额外扫描信号



扫描 RESET 引脚

RESET引脚接受5V电平的低有效逻辑电平以实现标准的复位操作；以及12V的高有效电平以实现高电压并行编程。Figure 127 所示的只能观测的扫描单元既适用于 5V 复位信号 RSTT，也适用于 12V 的复位信号 RSTHV。

Figure 127. 只能观测的单元



扫描时钟引脚

AVR器件可以通过熔丝位选择多种时钟选项，如片内RC振荡器、外部RC振荡器、外部时钟、（高频）晶体振荡器、低频晶体振荡器，以及陶瓷振荡器。

Figure 128 说明扫描链是如何支持每一种振荡器的。使能信号为普通的边界扫描单元，而振荡器 / 时钟的输出则连接到只可观测的单元。实时时钟振荡器的扫描方式与主时钟的一样。片内 RC 振荡器的输出是不能扫描的，因为这个振荡器没有外部连接。

Figure 128. 振荡器和时钟的边界扫描单元

Table 102简单总结了外部时钟引脚XTAL1、与XTAL1/XTAL2连接的振荡器，以及32 kHz 时钟振荡器的扫描寄存器。

Table 102. 振荡器的扫描信号 (1)(2)(3)

使能信号	扫描的时钟	时钟选项	未使用时扫描的时钟
EXTCLKEN	EXTCLK (XTAL1)	外部时钟	0
OSCON	OSCCK	外部晶体 外部陶瓷振荡器	0
RCOSCEN	RCCK	外部 RC	1
OSC32EN	OSC32CK	外部低频晶体振荡器	0
TOSKON	TOSCK	32 kHz 时钟振荡器	0

Notes:
1. 不要同时使能多于一个的时钟源作为主时钟。
2. 扫描振荡器输出可能得到不可预期的结果。这是因为振荡器和 JTAG TCK 时钟之间有频率漂移。扫描外部时钟更可行。
3. 时钟配置通过熔丝位进行。由于运行时不能改变熔丝位，因此对于一个应用可以认为时钟配置是固定的。建议用户在扫描时钟时使用与最终系统一样的选项。由于在睡眠模式下系统有可能禁止时钟，扫描链也支持时钟使能信号，从而将振荡器引脚从扫描链中断开。扫描链不支持熔丝位 INTCAP。所以边界扫描链无法使要求内部电容的 XTAL 振荡器运行，除非此熔丝位已经得到正确编程。

扫描模拟比较器

与边界扫描相关的比较器信号示于 Figure 129。 Figure 130 给出的边界扫描单元与这些信号相连接。信号在 Table 103 中说明。

对于纯粹的连接性测试可以避开比较器，因为所有的模拟输入与数字输入共享端口引脚。

Figure 129. 模拟比较器

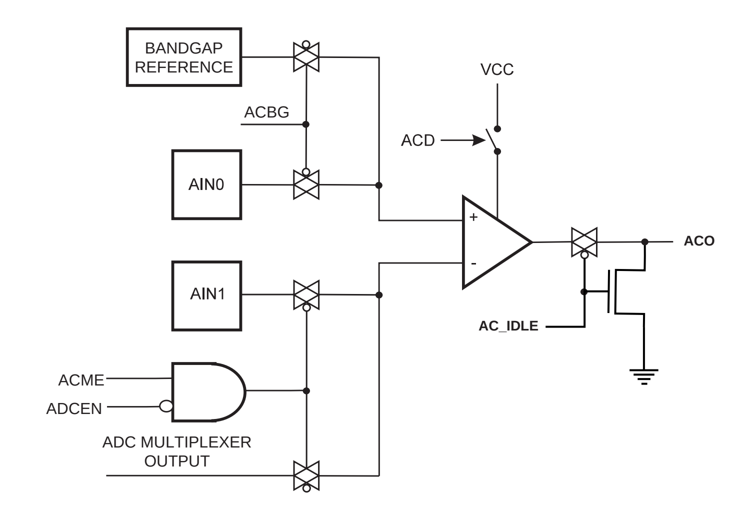


Figure 130. 适用于比较器和 ADC 的通用边界扫描单元

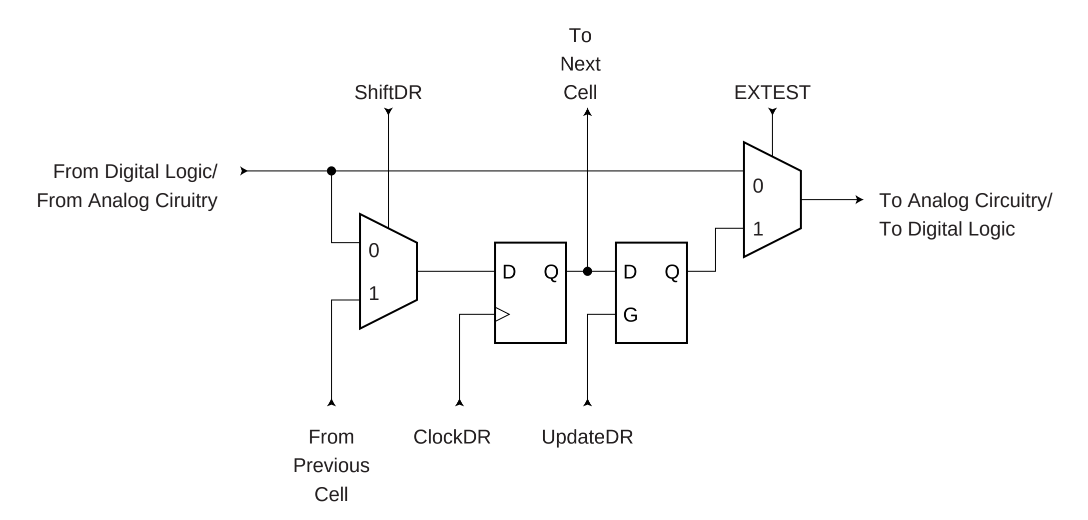


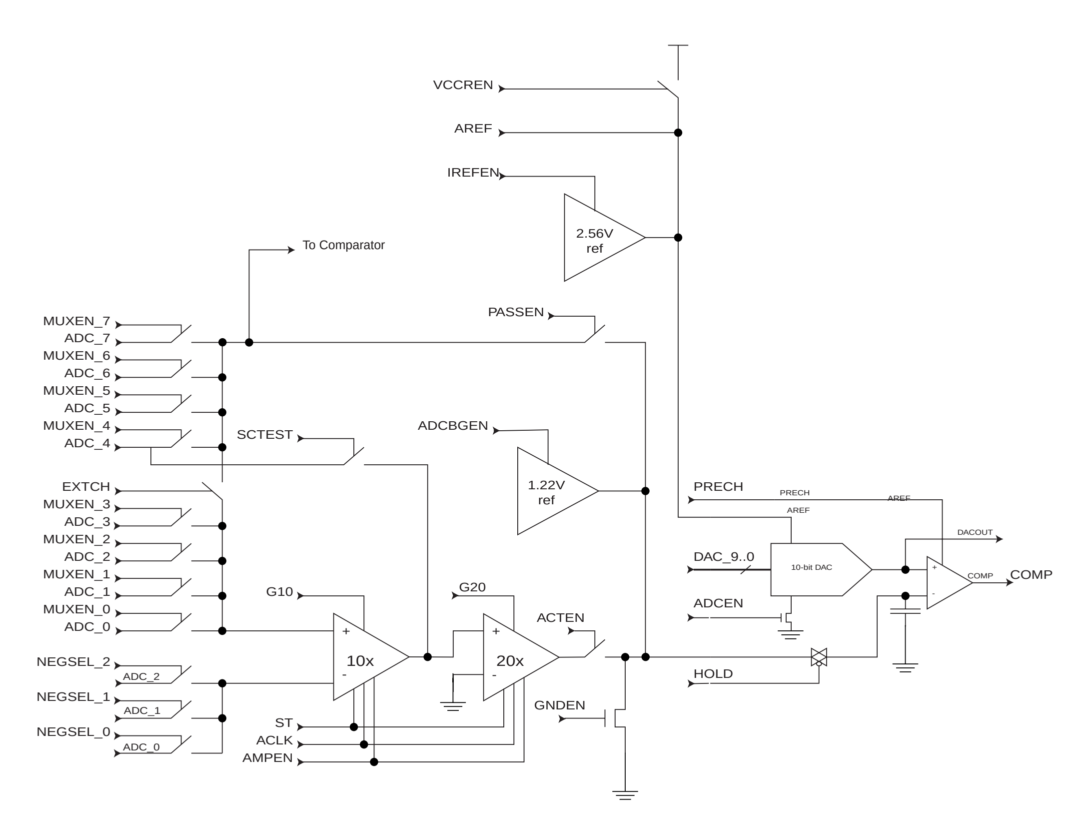
Table 103. 模拟比较器的边界扫描单元

信号名称	从比较器一侧看的方向	说明	不使用用时推荐的输入	使用推荐的输入时的输出数值
AC_IDLE	输入	为 '1' 时关闭模拟比较器	1	依赖于正在执行的 μC 代码
ACO	输出	模拟比较器的输出	将成为正在执行的 μC 代码的输入	0
ACME	输入	为 '1' 时使用来自 ADC 多路器的输出	0	依赖于正在执行的 μC 代码
ACBG	输入	使能能隙基准源	0	依赖于正在执行的 μC 代码

扫描 ADC

Figure 131 给出了具有所有相关的控制和观测信号的 ADC 原理图。Figure 127 所示的边界扫描单元与这些信号相连。对于纯粹的连接性测试可以避开 ADC，因为所有的模拟输入与数字输入共享端口引脚。

Figure 131. 模数转换器



信号在 Table 104 有简短描述。

Table 104. ADC 的边界扫描信号

信号名称	输出	比较器输出	不使用时推荐的输入	当推荐的输入正在使用，且 CPU 没有使用 ADC 时的输出数值
COMP	输入	以开关电容滤波器方式实现的增益的时钟信号	0	0
ACLK	输入	使能从增益到比较器的通路	0	0
ACTEN	输入	以消耗更大电流的方式提高比较器的速度	0	0
ADCBGEN	输入	使能能隙基准源连接到比较器的负输入端	0	0
ADCEN	输入	ADC 的上电信号	0	0
AMPEN	输入	增益的的上电信号	0	0
DAC_9	输入	DAC 数字数值的第 9 位	1	1
DAC_8	输入	DAC 数字数值的第 8 位	0	0
DAC_7	输入	DAC 数字数值的第 7 位	0	0
DAC_6	输入	DAC 数字数值的第 6 位	0	0
DAC_5	输入	DAC 数字数值的第 5 位	0	0
DAC_4	输入	DAC 数字数值的第 4 位	0	0
DAC_3	输入	DAC 数字数值的第 3 位	0	0
DAC_2	输入	DAC 数字数值的第 2 位	0	0
DAC_1	输入	DAC 数字数值的第 1 位	0	0
DAC_0	输入	DAC 数字数值的第 0 位	0	0
EXTCH	输入	将 ADC 的 0 - 3 通道连接到旁路通道	1	1
G10	输入	使能 10x 增益	0	0
G20	输入	使能 20x 增益	0	0
GNDEN	输入	为 '1' 时将比较器的负输入端连接到地	0	0
HOLD	输入	采样 & 保持信号。为 '0' 时采样模拟信号，为 '1' 时保持信号。如果使能增益，在 ACLK 为高时这个信号必须有效	1	1
IREFEN	输入	使能能隙基准源作为 DAC 的 AREF 信号	0	0
MUXEN_7	输入	输入多路器的位 7	0	0
MUXEN_6	输入	输入多路器的位 6	0	0
MUXEN_5	输入	输入多路器的位 5	0	0
MUXEN_4	输入	输入多路器的位 4	0	0
MUXEN_3	输入	输入多路器的位 3	0	0

Table 104. ADC 的边界扫描信号 (Continued)

信号名称	输出	比较器输出	不使用时推荐的输入	当推荐的输入正在使用，且 CPU 没有使用 ADC 时的输出数值
MUXEN_2	输入	输入多路器的位 2	0	0
MUXEN_1	输入	输入多路器的位 1	0	0
MUXEN_0	输入	输入多路器的位 0	1	1
NEGSEL_2	输入	差分信号负输入端的输入多路器的位 2	0	0
NEGSEL_1	输入	差分信号负输入端的输入多路器的位 1	0	0
NEGSEL_0	输入	差分信号负输入端的输入多路器的位 0	0	0
PASSEN	输入	使能增益的传输门	1	1
PRECH	输入	比较器输出锁存器预充电 (低有效)	1	1
SCTEST	输入	开关电容 TEST 使能。10x 增益输出发送到 ADC_4 引脚	0	0
ST	输入	如果此信号在 AMPEN 变高之后的头两个 ACLK 周期里为高，增益的输出将以更快的速度稳定下来	0	0
VCCREN	输入	选择 Vcc 为 ADC 的基准源	0	0

Note: 若不正确地设置 Figure 131 中的选项将造成信号冲突，并有可能毁坏器件。Figure 131 中输出比较器负输入端的 S&H 电路的输出有多种选择。要保证在 ADC 引脚、能隙基准源和地三者之一选择一个。

若在扫描过程中没有使用 ADC，则推荐使用 Table 104 给出的输入值。不推荐用户在扫描过程中使用差分增益。基于开关电容的增益需要快速的操作和精确的定时，这在使用扫描链时是很难保证的。出于同样的原因， ADC 高速模式 (ADHSM) 也不推荐在边界扫描时使用。

AVR ADC 的模拟电路示于 Figure 131，是以数字逻辑实现的逐次逼近算法。使用边界扫描的问题是保证施加的模拟电压在某些限制下得以测量。这可以方便地通过不运行逐次逼近算法来实现：在数字 DAC[9:0] 上施加下限值，确保比较器的输出为低；然后在数字 DAC[9:0] 上施加上限值，确保比较器的输出为高。

对于纯粹的连接性测试可以避开 ADC，因为所有的模拟输入与数字输入共享端口引脚。

使用 ADC 时请记住

- 使用 ADC 时必须将引脚配置为输入，而且要禁止上拉电阻，以防止信号冲突。
- 在正常模式下，使能ADC时将启动一次"哑“模数转换(包括10次比较)。建议用户在使能 ADC 后等待至少 200ns，然后再开始操作；或者是执行一次 " 哑 “ 模数转换。
- HOLD 信号为低时 (采样模式)， DAC 数值必须稳定于中间值 0x200。



作为例子，考虑电源电压为 5.0V、 AREF 连接到 V_{CC} 时在 ADC 通道 3 施加 1.5V ± 5% 的输入信号 。

The lower limit is: $\lceil 1024 \cdot 1{,}5V \cdot 0{,}95 / 5V \rceil = 291 = 0x123$
The upper limit is: $\lceil 1024 \cdot 1{,}5V \cdot 1{,}05 / 5V \rceil = 323 = 0x143$

除非使用 Table 105 所示算法的数值，建议使用 Table 104 的推荐数据。 Table 105 只给出了扫描链中 DAC 和端口引脚的数值。“ 动作 ” 一列说明了在填充后面几列边界扫描寄存器之前要使用的 JTAG 指令。在移入数据的时候要对移出的数据进行校验。

Table 105. 使用 ADC 的算法

步骤	动作	ADCEN	DAC	MUXEN	HOLD	PRECH	PA3. 数据	PA3. 控制	PA3. 上拉使能
1	SAMPLE_ PRELOAD	1	0x200	0x08	1	1	0	0	0
2	EXTEST	1	0x200	0x08	0	1	0	0	0
3		1	0x200	0x08	1	1	0	0	0
4		1	0x123	0x08	1	1	0	0	0
5		1	0x123	0x08	1	0	0	0	0
6	校验 COMP 是否为 0	1	0x200	0x08	1	1	0	0	0
7		1	0x200	0x08	0	1	0	0	0
8		1	0x200	0x08	1	1	0	0	0
9		1	0x143	0x08	1	1	0	0	0
10		1	0x143	0x08	1	0	0	0	0
11	校验 COMP 是否为 1	1	0x200	0x08	1	1	0	0	0

使用这个算法时对 HOLD 信号的时序约束将对 TCK 时钟产生约束。由于算法在 5 个步骤里将 HOLD 保持为高， TCK 时钟必须至少 5 倍于扫描位数除以最大保持时间 $t_{hold,max}$。

ATmega128 边界扫描次序

选择边界扫描链为数据通路时 TDI 和 TDO 之间的扫描次序如 Table 106 所示。 Bit 0 为 LSB，是第一个被扫描 (输出 / 输入) 的位。按照引脚次序进行扫描是不可能的。因此，端口 A 的扫描次序与其他端口是相反的。模拟电路的扫描链是一个例外，它构成了扫描链的最高位，而不管连接到哪一个物理引脚。在 Figure 124 里， PXn.Data 相应于 FF0 ； PXn.Control 相应于 FF1 ； PXn. Pullup_enable 相应于 FF2。端口 C 的 2、3、4、5 位不在扫描链之中，因为 JTAG 使能时这些引脚是 TAP 引脚。

Table 106. ATmega128 边界扫描次序

各个位的序号	信号名称	模块
204	AC_IDLE	比较器
203	ACO	
202	ACME	
201	AINBG	
200	COMP	ADC
199	PRIVATE_SIGNAL1[(1)]	
198	ACLK	
197	ACTEN	
196	PRIVATE_SIGNAL1[(2)]	
195	ADCBGEN	
194	ADCEN	
193	AMPEN	
192	DAC_9	
191	DAC_8	
190	DAC_7	
189	DAC_6	
188	DAC_5	
187	DAC_4	
186	DAC_3	
185	DAC_2	
184	DAC_1	
183	DAC_0	
182	EXTCH	
181	G10	
180	G20	
179	GNDEN	
178	HOLD	
177	IREFEN	
176	MUXEN_7	



Table 106. ATmega128 边界扫描次序

各个位的序号	信号名称	模块
175	MUXEN_6	ADC
174	MUXEN_5	
173	MUXEN_4	
172	MUXEN_3	
171	MUXEN_2	
170	MUXEN_1	
169	MUXEN_0	
168	NEGSEL_2	
167	NEGSEL_1	
166	NEGSEL_0	
165	PASSEN	
164	PRECH	
163	SCTEST	
162	ST	
161	VCCREN	
160	PEN	编程使能 (只可观测)
159	PE0.Data	端口 E
158	PE0.Control	
157	PE0.Pullup_Enable	
156	PE1.Data	
155	PE1.Control	
154	PE1.Pullup_Enable	
153	PE2.Data	
152	PE2.Control	
151	PE2.Pullup_Enable	
150	PE3.Data	
149	PE3.Control	
148	PE3.Pullup_Enable	
147	PE4.Data	
146	PE4.Control	
145	PE4.Pullup_Enable	
144	PE5.Data	
143	PE5.Control	
142	PE5.Pullup_Enable	
141	PE6.Data	
140	PE6.Control	

Table 106. ATmega128 边界扫描次序

各个位的序号	信号名称	模块
139	PE6.Pullup_Enable	端口 E
138	PE7.Data	
137	PE7.Control	
136	PE7.Pullup_Enable	
135	PB0.Data	端口 B
134	PB0.Control	
133	PB0.Pullup_Enable	
132	PB1.Data	
131	PB1.Control	
130	PB1.Pullup_Enable	
129	PB2.Data	
128	PB2.Control	
127	PB2.Pullup_Enable	
126	PB3.Data	
125	PB3.Control	
124	PB3.Pullup_Enable	
123	PB4.Data	
122	PB4.Control	
121	PB4.Pullup_Enable	
120	PB5.Data	
119	PB5.Control	
118	PB5.Pullup_Enable	
117	PB6.Data	
116	PB6.Control	
115	PB6.Pullup_Enable	
114	PB7.Data	
113	PB7.Control	
112	PB7.Pullup_Enable	
111	PG3.Data	端口 G
110	PG3.Control	
109	PG3.Pullup_Enable	
108	PG4.Data	
107	PG4.Control	
106	PG4.Pullup_Enable	
105	TOSC	32 kHz 时钟振荡器
104	TOSCON	



Table 106. ATmega128 边界扫描次序

各个位的序号	信号名称	模块
103	RSTT	复位逻辑 （只可观测）
102	RSTHV	
101	EXTCLKEN	主时钟 / 振荡器的只能信号
100	OSCON	
99	RCOSCEN	
98	OSC32EN	
97	EXTCLK (XTAL1)	主时钟的时钟输入和振荡器 （只可观测）
96	OSCCK	
95	RCCK	
94	OSC32CK	
93	TWIEN	TWI
92	PD0.Data	端口 D
91	PD0.Control	
90	PD0.Pullup_Enable	
89	PD1.Data	
88	PD1.Control	
87	PD1.Pullup_Enable	
86	PD2.Data	
85	PD2.Control	
84	PD2.Pullup_Enable	
83	PD3.Data	
82	PD3.Control	
81	PD3.Pullup_Enable	
80	PD4.Data	
79	PD4.Control	
78	PD4.Pullup_Enable	
77	PD5.Data	
76	PD5.Control	
75	PD5.Pullup_Enable	
74	PD6.Data	
73	PD6.Control	
72	PD6.Pullup_Enable	
71	PD7.Data	
70	PD7.Control	
69	PD7.Pullup_Enable	
68	PG0.Data	端口 G

Table 106. ATmega128 边界扫描次序

各个位的序号	信号名称	模块
67	PG0.Control	端口 G
66	PG0.Pullup_Enable	
65	PG1.Data	
64	PG1.Control	
63	PG1.Pullup_Enable	
62	PC0.Data	端口 C
61	PC0.Control	
60	PC0.Pullup_Enable	
59	PC1.Data	
58	PC1.Control	
57	PC1.Pullup_Enable	
56	PC2.Data	
55	PC2.Control	
54	PC2.Pullup_Enable	
53	PC3.Data	
52	PC3.Control	
51	PC3.Pullup_Enable	
50	PC4.Data	
49	PC4.Control	
48	PC4.Pullup_Enable	
47	PC5.Data	
46	PC5.Control	
45	PC5.Pullup_Enable	
44	PC6.Data	
43	PC6.Control	
42	PC6.Pullup_Enable	
41	PC7.Data	
40	PC7.Control	
39	PC7.Pullup_Enable	
38	PG2.Data	端口 G
37	PG2.Control	
36	PG2.Pullup_Enable	
35	PA7.Data	端口 A
34	PA7.Control	
33	PA7.Pullup_Enable	
32	PA6.Data	





Table 106. ATmega128 边界扫描次序

各个位的序号	信号名称	模块
31	PA6.Control	端口 A
30	PA6.Pullup_Enable	
29	PA5.Data	
28	PA5.Control	
27	PA5.Pullup_Enable	
26	PA4.Data	
25	PA4.Control	
24	PA4.Pullup_Enable	
23	PA3.Data	
22	PA3.Control	
21	PA3.Pullup_Enable	
20	PA2.Data	
19	PA2.Control	
18	PA2.Pullup_Enable	
17	PA1.Data	
16	PA1.Control	
15	PA1.Pullup_Enable	
14	PA0.Data	
13	PA0.Control	
12	PA0.Pullup_Enable	
11	PF3.Data	端口 F
10	PF3.Control	
9	PF3.Pullup_Enable	
8	PF2.Data	
7	PF2.Control	
6	PF2.Pullup_Enable	
5	PF1.Data	
4	PF1.Control	
3	PF1.Pullup_Enable	
2	PF0.Data	
1	PF0.Control	
0	PF0.Pullup_Enable	

Notes: 1. PRIVATE_SIGNAL1 总是扫描为 0。
2. PRIVATE_SIGNAL2 总是扫描为 0。

边界扫描描述语言文件

边界扫描描述语言 (BSDL) 文件以一种为自动化测试生成软件使用的标准格式描述具有边界扫描功能的器件。这种文件的使用对象为自动化的测试生成软件。边界扫描数据寄存器各个位的次序和功能都包含于此文件中。



支持引导装入程序 – 在写的同时可以读 (RWW, Read-While-Write) 的自我编程能力

Boot Loader为通过MCU本身来下载和上载程序代码提供了一个真正的同时读-写(Read-While-Write ,以下简称 RWW) 自编程机制。这一特点使得系统可以在 MCU 的控制下 ,通过驻留于程序 Flash 的 Boot Loader，灵活地进行应用软件升级。 Boot Loader 可以使用任何器件具有的数据接口和相关的协议获得代码并把代码（程序）写入 Flash，或者从程序存储器读取代码。Boot Loader 区的程序可以写整个 Flash ,包括 Boot Loader 区本身。因而 Boot Loader 可以对其自身进行修改，甚至将自己擦除。Boot Loader 存储器空间的大小可以通过熔丝位进行配置。 Boot Loader 具有两套程序加密位，各自可以独立设置，给用户提供了选择保护级的灵活性。

引导程序的特点

- **RWW 自编程**
- **灵活的 Boot Loader 存储区配置**
- **高度的安全性 (有单独的 Boot 锁定位实现灵活的程序保护)**
- **有独立的熔丝位用于选择复位向量**
- **优化的页 [(1)] 大小**
- **代码优化的算法**
- **有效的 RWW 支持**

Note: 1. 页是 Flash 的组成部分 , 由数个字节组成 (见 P 273Table 123) , 在编程过程中使用。页的组织结构不影响正常的操作。

应用程序 Flash 区以及引导程序 Flash 区

Flash 由两个区构成 , 应用区和 Boot Loader 区 (见 Figure 133)。两个区的存储空间大小由 BOOTSZ 熔丝位配置，如 P 266Table 和 Figure 133 所示。由于两个区使用不同的锁定位，所以可以具有不同的加密级别。

应用程序区

应用区是 Flash 用来存储应用代码的区域。应用区的保护级别通过应用 Boot 锁定位 (Boot 锁定位 0) 确定 , 详见 P 258Table 。由于 SPM 指令在应用区执行时是无效的 , 所以应用区不能用来存储 Boot Loader 代码。

引导程序区 (Boot Loader Section) - BLS

应用区用来存储应用代码，而 Boot Loader 软件必须保存在 BLS。这是因为只有在 BLS 运行时 SPM 指令才有效。SPM 指令可以访问整个 Flash , 包括 BLS 本身。Boot Loader 区的保护级别通过 Boot Loader 锁定位 (Boot 锁定位 1) 确定，详见 P 259Table 109 。

RWW Flash 区及非 RWW Flash 区

CPU 是否支持 RWW , 或者 CPU 是否在利用 Boot Loader 软件进行代码更新时停止 , 取决于被编程的是哪个地址。除了前面所述的通过 BOOTSZ 熔丝位配置的两个区之外 , Flash 还可以分成两个固定的区——同时读 - 写 (RWW) 区和非同时读 - 写 (NRWW) 区。RWW- 和 NRWW 的分界在 P 266Table Note: 和 P 258Figure 133 给出。两个区的主要区别是 :

- 对 RWW 区内的页进行擦除或写操作时可以读 NRWW 区。
- 对 NRWW 区内的页进行擦除或写操作时， CPU 停止。

注意，Boot Loader 软件工作时 , 用户软件不能读取位于 RWW 区内的任何代码。"RWW 区 " 指的是被编程（擦除或写）的那个存储区，而不是利用 Boot Loader 软件进行代码更新过程中实际被读取的那部分。

RWW 区

如果 Boot Loader 软件是对 RWW 区内的某一页进行编程 , 则可以从 Flash 中读取代码，但只限于 NRWW 区内的代码。在 Flash 编程期间 , 用户软件必须保证没有对 RWW 区的读访问。如果用户软件在编程过程中试图读取位于 RWW 区的代码（如通过 call/jmp/lpm 指令或中断），软件可能会终止于一个未知状态。为了避免这种情况的发生，需要禁止中断或将其转移到 Boot Loader 区。 Boot Loader 总是位于 NRWW 存储区。只要 RWW 区处于不能读访问的状态，存储程序存储器控制和状态寄存器 (SPMCSR) 的 RWW 区忙标志位 RWWSB 置位。编程结束后 , 要在读取 位于 RWW 区的代码之前通过软件清除 RWWSB。具体如何清除 RWWSB 请参见 P 258 “ 保存程序存储器控制寄存器 - SPMCSR” 。

非 RWW 区 - NRWW

在 Boot Loader 软件更新 RWW 区的某一页时，可以读取位于 NRWW 区的代码。当 Boot Loader 代码更新 NRWW 区时，在整个页擦除或写操作过程中 CPU 被挂起。

Table 107. RWW 的特点

编程过程中 Z 指针寻址哪个区？	编程过程中可以读取哪个区？	CPU 挂起吗？	支持 RWW 吗？
RWW 区	NRWW 区	不	是
NRWW 区	无	是	不

Figure 132. RWW 与 NRWW

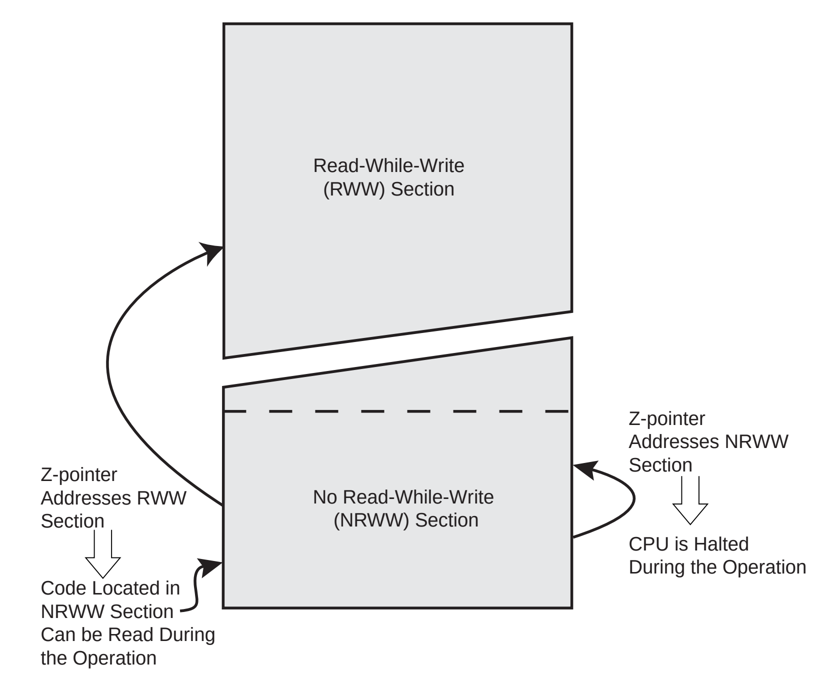
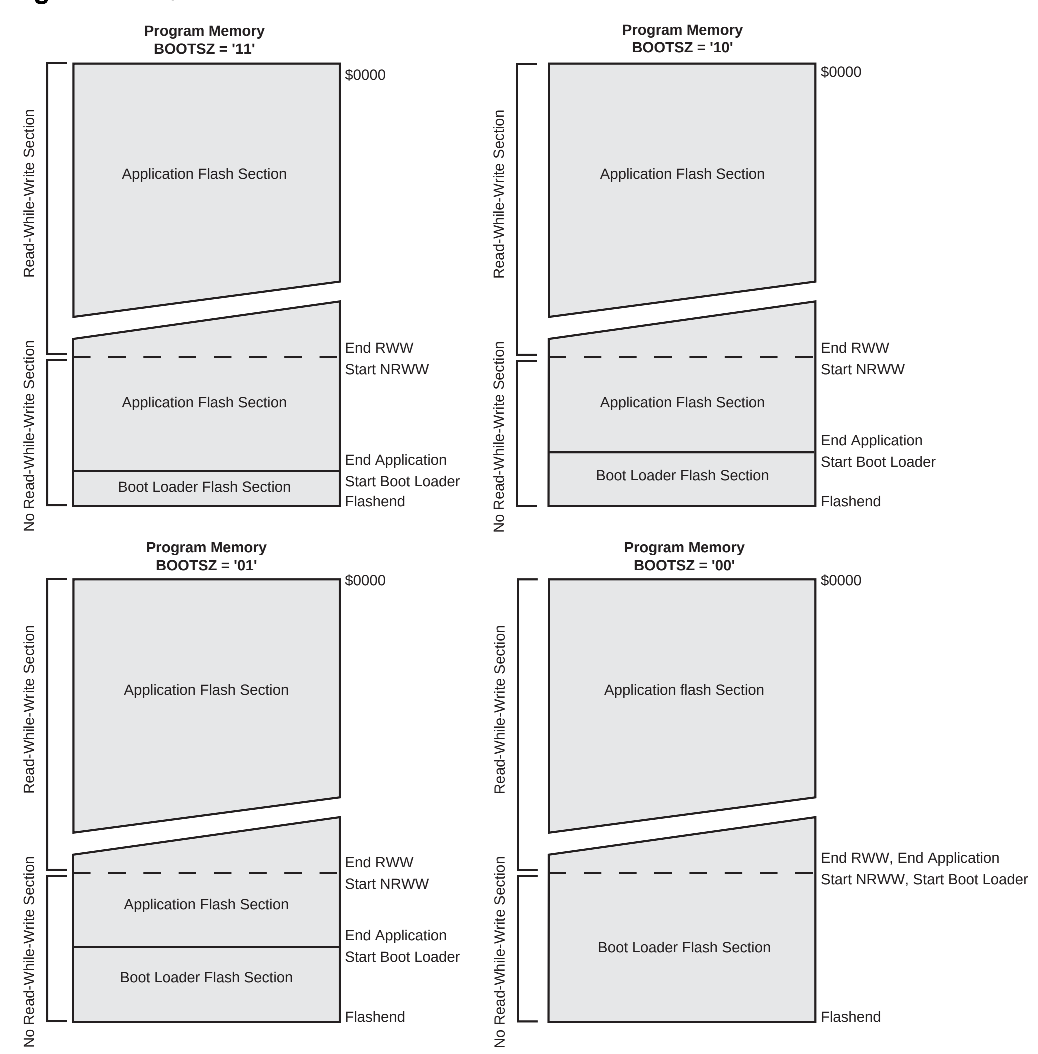


Figure 133. 存储器区 (1)



Note: 1. 上图中的参数在 P 266Table 中给出。

引导程序区锁定位

如果不需要 Boot Loader 功能，则整个 Flash 都可以为应用代码所用。Boot Loader 具有两套可以独立设置的 Boot 锁定位。用户可以灵活地选择不同的代码保护方式。

用户可以选择：

- 保护整个 Flash 区，不让 MCU 进行软件升级。
- 不允许 MCU 升级 Boot Loader Flash 区。
- 不允许 MCU 升级应用 Flash 区。
- 允许 MCU 升级整个 Flash 区。

详细内容请参见 Table 108 与 Table 109。 Boot 锁定位可以通过软件、串行下载或并行编程进行设置，但只能通过芯片擦除命令清除。通用的写锁定位 (锁定位模式 2) 不限制通过 SPM 指令对 Flash 进行编程。与此类似，通用的读 / 写锁定位 (锁定位模式 1) 也不限制通过 LPM/SPM 指令对闪存进行读 / 写访问。

Table 108. Boot 锁定位 0 保护模式 (应用区)[1]

BLB0 模式	BLB02	BLB01	保护
1	1	1	允许 SPM/LPM 指令访问应用区
2	1	0	不允许 SPM 指令对应用区进行写操作
3	0	0	不允许 SPM 指令对应用区进行写操作，也不允许运行于 Boot Loader 区的 LPM 指令从应用区读取数据。若中断向量位于 Boot Loader 区，那么执行应用区代码时中断是禁止的。
4	0	1	不允许运行于 Boot Loader 区的 LPM 指令从应用区读取数据。若中断向量位于 Boot Loader 区，那么执行应用区代码时中断是禁止的。

Note: 1. “1” 表示未编程，“0” 表示已编程。

Table 109. Boot 锁定位 1 保护模式 (Boot Loader 区)[1]

BLB1 模式	BLB12	BLB11	保护
1	1	1	允许 SPM/LPM 指令访问 Boot Loader 区
2	1	0	不允许 SPM 指令对 Boot Loader 区进行写操作
3	0	0	不允许 SPM 指令对 Boot Loader 区进行写操作，也不允许运行于应用区的 LPM 指令从 Boot Loader 区读取数据。若中断向量位于应用区，那么执行 Boot Loader 区代码时中断是禁止的。
4	0	1	不允许运行于应用区的 LPM 指令从 Boot Loader 区读取数据。若中断向量位于应用区，那么执行 Boot Loader 区代码时中断是禁止的。

Note: 1. “1” 表示未编程，“0” 表示已编程。

进入引导程序

通过跳转指令或从应用区调用的方式可以进入 Boot Loader。这些操作可以由一些触发信号启动，比如通过 USART 或 SPI 接口接收到了相关的命令。另外，可以通过编程 Boot 复位熔丝位使得复位向量指向 Boot 区的起始地址。这样，复位后 Boot Loader 立即就启动了。加载了应用代码后，程序开始执行应用代码。 MCU 本身不能改变熔丝位的设置。也就是说，一旦 Boot 复位熔丝位被编程，复位向量将一直指向 Boot 区的起始地址。熔丝位只能通过串行或并行编程的方法来改变。

Table 110. Boot 复位熔丝位 [1]

BOOTRST	复位地址
1	Reset Vector = 应用区复位 (地址 $0000)
0	Reset Vector = Boot Loader 复位 (见 P 266Table 112)

Note: 1. “1” 意味着未编程 , “0” 意味着已编程。

保存程序存储器控制寄存器 - SPMCSR

存贮程序存储控制器和状态寄存器包括了控制 Boot Loader 操作所需的控制位。

Bit	7	6	5	4	3	2	1	0	
	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	SPMCSR
读 / 写	R/W	R	R	R/W	R/W	R/W	R/W	R/W	
初始值	0	0	0	0	0	0	0	0	

- **Bit 7 – SPMIE: SPM 中断使能**

SPMIE 置位后，如果状态寄存器的 I 位也置位，SPM 中断即被使能。只要 SPMCSR 寄存器的 SPMEN 清零， SPM 中断将被执行。

- **Bit 6 – RWWSB:RWW 区忙标志**

启动对 RWW 区的自编程（页擦除或页写入）操作时， RWWSB 被硬件置 1。 RWWSB 置位时不能访问 RWW 区。自编程操作完成后，如果 RWWSRE 位为 1， RWWSB 位将被清除。另外，启动页加载操作将使 RWWSB 位自动清零。

- **Bit 5 – Res: 保留位**

在 ATmega128 中为保留位，读返回值为 "0"。

- **Bit 4 – RWWSRE: RWW 区读使能**

RWW区处于编程(页擦除或页写入)状态时，RWW区的读操作(RWWSB被硬件置"1")将被阻塞。用户软件必须等到编程结束(SPMEN清零)才能重新使能RWW区。如果 RWWSRE 位和SPMEN同时被写入"1"，则在紧接着的四个时钟周期内的SPM指令将再次使能RWW区。如果 Flash 忙于页擦除或页写入 (SPMEN 置位),RWW 区不能被使能。如果 Flash 加载与 RWWSRE 写操作同时发生，则 Flash 加载操作终止，加载的数据亦将丢失。

- **Bit 3 – BLBSET: Boot 锁定位设置**

如果这一位和 SPMEN 同时置位，发生于紧接着的四个时钟周期内的 SPM 指令会根据 R0 中的数据设置 Boot 锁定位。R1 中的数据和 Z 指针的地址信息被忽略。锁定位设置完成，或在四个时钟周期内没有 SPM 指令被执行时， BLBSET 自动清零。

在 SPMCSR 寄存器的 BLBSET 和 SPMEN 置位后的三个周期内运行的 LPM 指令将读取锁定位或熔丝位（取决于 Z 指针的 Z0) 并送到目的寄存器。详见 P 262“ 以软件方式读取熔丝位和锁定位 ” 。

- **Bit 2 – PGWRT: 页写入**

如果这一位和 SPMEN 同时置位，发生于紧接着的四个时钟周期内的 SPM 指令执行页写功能，将临时缓冲器中存储的数据写入 Flash。页地址取自 Z 指针的高位部分。R1 和 R0 的数据则被忽略。页写操作完成，或在四个时钟周期内没有 SPM 指令被执行时，PGWRT 自动清零。如果页写对象为 NRWW 区，在整个页写操作过程中 CPU 停止。

- **Bit 1 – PGERS: 页擦除**

如果这一位和 SPMEN 同时置位，发生于紧接着的四个时钟周期内的 SPM 指令执行页擦除功能。页地址取自 Z 指针的高位部分。 R1 和 R0 的数据则被忽略。页擦除操作完成，或在四个时钟周期内没有SPM指令被执行时，PGERS自动清零。如果页写对象为NRWW 区，在整个页擦除操作过程中 CPU 停止。

- **Bit 0 – SPMEN: 存贮程序存储器使能**

这一位使能紧接着的四个时钟周期内的 SPM 指令。如果将这一位和 RWWSRE、BLBSET、PGWRT 或 PGERS 之一同时置位，则如上所述，接下来的 SPM 指令将有特殊的含义。如果只有 SPMEN 置位，那么接下来的 SPM 指令将把 R1:R0 中的数据存储到由 Z 指针确定的临时页缓冲器。Z 指针的 LSB 被忽略。SPM 指令完成，或在四个时钟周期内没有 SPM 指令被执行时， SPMEN 自动清零。在页擦除和页写过程中 SPMEN 保持为 1 直到操作完成。

在低五位中写入除 “10001”、“01001”、“00101”、 “00011” 或 “00001” 之外的任何组合都无效。

在自编程时访问 Flash

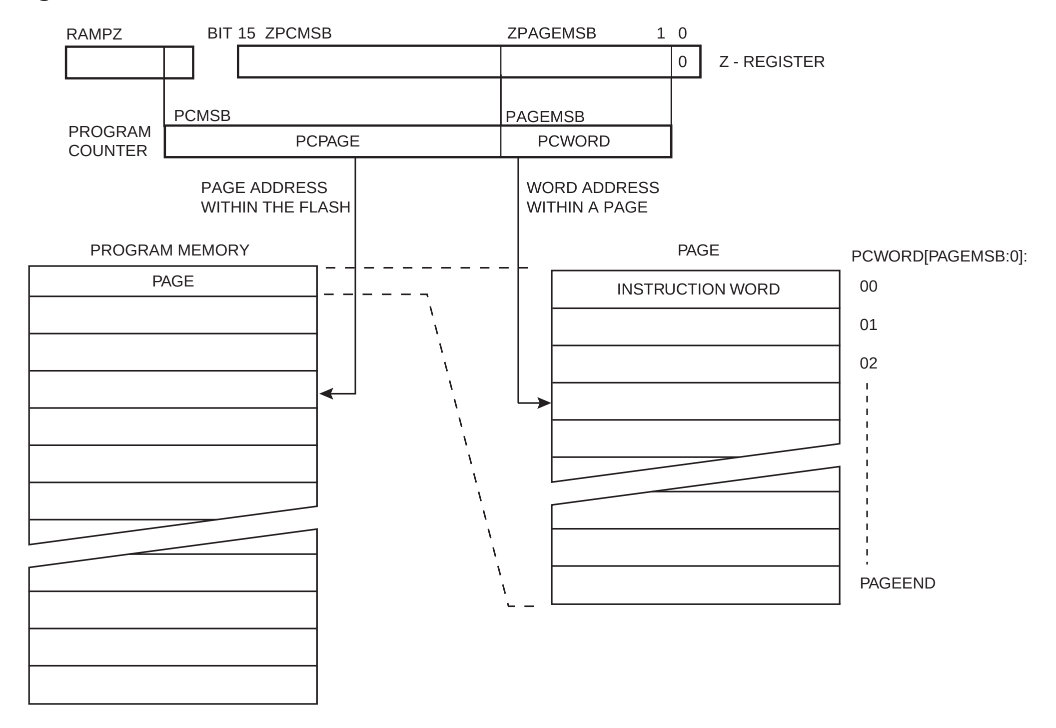
Z指针与RAMPZ用于SPM命令的寻址。如何使用RAMPZ，详见 P 11 “RAM页面的Z选择寄存器 - RAMPZ” 。

Bit	15	14	13	12	11	10	9	8
ZH (R31)	Z15	Z14	Z13	Z12	Z11	Z10	Z9	Z8
ZL (R30)	Z7	Z6	Z5	Z4	Z3	Z2	Z1	Z0
	7	6	5	4	3	2	1	0

由于 Flash 存储器是以页的形式组织 (P 273Table 123) 起来的，程序计数器可看作由两个部分构成 : 其一为实现页内寻址的低位部分；其次为实现页寻址的高位部分，如 Figure 134所示。由于页擦除和页写操作的寻址是相互独立的 ,因此保证Boot Loader软件在页擦除和页写操作时寻址相同的页是最重要的。一旦编程操作开始启动，地址就被锁存，然后 Z 指针 /RAMPZ 可以用作其他用途了。

唯一不使用 Z 指针 /RAMPZ 的 SPM 操作是设置 Boot Loader 锁定位。 Z 指针 /RAMPZ 的内容被忽略。 (E)LPM 指令也使用 Z 指针 /RAMPZ 来保存地址。由于这个指令的寻址逐字节地进行，所以 Z 指针的 LSB 位 (位 Z0) 也使用到了。

Figure 134. SPM 的寻址 (1)



Notes: 1. Figure 134 中所用的不同的变量在 P 267Table 114 列出。
2. PCPAGE 与 PCWORD 列于 P 273Table 124 。

Flash 的自编程

程序存储器的更新以页的方式进行。在用临时页缓冲器存储的数据对一页存储器进行编程时，首先要将这一页擦除。 SPM 指令以一次一个字的方式将数据写入临时页缓冲器。临时页缓冲器的写入可以在页擦除命令之前完成，也可以在页擦除和页写操作之间完成。

方案 1，在页擦除前写缓冲器 :

- 写临时页缓冲器
- 执行页擦除操作
- 执行页写操作

方案 2，在页擦除后写缓冲器 :

- 执行页擦除操作
- 写临时页缓冲器
- 执行页写操作

如果只需要改变页的一部分，则在页擦除之前必须将页中其他部分存储起来 (如保存于临时页缓冲区中)，然后再写回 Flash。使用方案 1 时， Boot Loader 提供了一个有效的读 - 修改 - 写特性，允许用户软件首先读取页中的内容，然后对内容做必要的改变，接着把修



改后的数据写回 Flash。如果使用方案 2，则无法读取旧数据，因为页已经被擦除了。临时页缓冲区可以随机寻址。保证在页擦除和页写操作中寻址相同的页是很关键的。汇编代码的例子请参见 P 263“ 一个简单的引导程序汇编代码 ”。

通过 SPM 执行页擦除

执行页擦除操作首先需要设置 Z 指针与 RAMPZ 的地址信息，然后将 “X0000011” 写入 SPMCSR，最后在其后的四个时钟周期内执行 SPM。R1 和 R0 中的数据被忽略。页地址必须写入 Z 寄存器的 PCPAGE。 Z 指针的其他位被忽略。

- 擦除 RWW 区的页：在页擦除过程中可以读取 NRWW 区。
- 擦除 NRWW 区的页：在操作过程中 CPU 停止。

装载临时缓冲器 (页加载)

写一个指令字首先需要设置 Z 指针的地址信息，以及将指令字写入 R1:R0，然后将 “00000001” 写入 SPMCSR，最后在其后的四个时钟周期内执行 SPM。 Z 寄存器中 PCWORD 的内容用来寻址临时缓冲区。页写操作完成，或置位 SPMCSR 寄存器的 RWWSRE 将使临时缓冲区自动擦除。系统复位也会擦除临时缓冲区。但是如果不清除临时缓冲区就只能对每个地址进行一次写操作。

Note: 若在 SPM 页载入操作中 EEPROM 写入，所有载入数据将丢失。

执行页写操作

执行页写操作首先需要设置 Z 指针与 RAMPZ 的地址信息，然后将 “X0000101” 写入 SPMCSR，最后在其后的四个时钟周期内执行 SPM。R1 和 R0 中的数据被忽略。页地址必须写入 Z 寄存器的 PCPAGE。 Z 指针的其他位被忽略。

- 擦除 RWW 区的页：在页擦除过程中可以读取 NRWW 区。
- 擦除 NRWW 区的页：在页写过程中 CPU 停止。

使用 SPM 中断

如果 SPM 中断使能，则 SPMCSR 寄存器的 SPMEN 清零将产生中断。这意味着软件可以利用中断来代替对 SPMCSR 寄存器的查询。使用 SPM 中断时，要将中断向量移到 BLS，以避免 RWW 区读禁止时中断程序却访问它。如何移动中断向量请见 P 55“ 中断 ”。

在更新 BLS 时需要考虑的问题

通过不编程 Boot 锁定位 11 的方式来更新 Boot Loader 区时需要给予格外关注。对 Boot Loader 本身进行的误操作会破坏整个 Boot Loader ，造成软件无法更新。如果程序不需要改变 Boot Loader，建议对 Boot 锁定位 11 编程，以防止不小心改变了 Boot Loader。

在自编程时防止读取 RWW 区

在自编程过程中 (页擦除或页写)，对 RWW 区的访问被阻塞。用户软件要避免此情况发生。RWW 区忙将使 SPMCSR 寄存器的 RWWSB 置位。在自编程时，如 P 55“ 中断 ” 所述，中断向量表应该移到 BLS 中，或者禁止中断。编程结束后，在寻址 RWW 区之前用户软件必须对 RWWSRE 写 1 来清零 RWWSB。例子请见 P 263“ 一个简单的引导程序汇编代码 ”。

通过 SPM 设置引导程序锁定位

设置Boot Loader 锁定位首先要给R0赋予期望的数据 ,然后将“X0001001”写入SPMCSR 寄存器，并在紧接着的四个时钟周期内执行 SPM 指令。唯一可访问的锁定位是 Boot Loader 锁定位。利用这个锁定位可以阻止 MCU 对应用程序和 Boot Loader 软件的更新。

Bit	7	6	5	4	3	2	1	0
R0	1	1	BLB12	BLB11	BLB02	BLB01	1	1

不同的 Boot Loader 锁定位设置对 Flash 访问的影响请参见 Table 108 和 Table 109。

如果 R0 的 5..2 位为 0 ,并且在 SPMCSR 寄存器的 BLBSET 和 SPMEN 置位之后的四个周期内执行了 SPM 指令，相应的 Boot 锁定位将被编程。此操作不使用 Z 指针，但出于兼容性的考虑 ,建议将 Z 指针赋值为 0x0001(与读 IO_{ck} 位的操作相同)。同样出于兼容性的考虑，建议在写锁定位时将 R0 中的 7、6、 1 和第 0 位置 "1"。在编程锁定位的过程中可以自由访问整个 Flash 区。

写 EEPROM 将阻止写 SPMCSR

EEPROM 写操作会阻塞对 Flash 的编程 , 也会阻塞对熔丝位和锁定位的读操作。建议用户在对 SPMCSR 寄存器进行写操作之前首先检查 EECR 寄存器的状态位 EEWE，确保此位以被清除。

以软件方式读取熔丝位和锁定位

熔丝位和锁定位可以通过软件读取。读锁定位时，需要将 $0001 赋予给 Z 指针并且置位 SPMCSR寄存器的BLBSET 和 SPMEN。在SPMCSR操作之后的三个CPU周期内执行的 LPM 指令将把锁定位的值将加载到目的寄存器。读锁定位操作结束 ,或者在三个 CPU 周期内没有 执行LPM指令 ,或在四个CPU周期内没有执行SPM指令 ,BLBSET 和 SPMEN 位将自动硬件清零。 BLBSET 和 SPMEN 清零后，LPM 将按照指令手册中所描述的那样工作。

Bit	7	6	5	4	3	2	1	0
Rd	–	–	BLB12	BLB11	BLB02	BLB01	LB2	LB1

读取熔丝位低字节的算法和上述读取锁定位的算法类似。要读取熔丝位低字节，需要将 $0000赋予给Z指针并且置位SPMCSR寄存器的BLBSET 和 SPMEN。在SPMCSR操作之后的三个 CPU 周期内执行的 LPM 指令将把熔丝位低位字节的值 (FLB) 加载到目的寄存器。更详细的说明及熔丝位低位字节映射的细节请参见 P 270Table 119 。

Bit	7	6	5	4	3	2	1	0
Rd	FLB7	FLB6	FLB5	FLB4	FLB3	FLB2	FLB1	FLB0

类似的 , 读取熔丝位高位字节时 , 需要将 $0003 赋予给 Z 指针并且置位 SPMCSR 寄存器的 BLBSET 和 SPMEN。在 SPMCSR 操作之后的三个 CPU 周期内执行的 LPM 指令将把熔丝位高位字节的值 (FHB) 加载到目的寄存器。更详细的说明及熔丝位高位字节映射的细节请参见 P 270Table 118 。

Bit	7	6	5	4	3	2	1	0
Rd	FHB7	FHB6	FHB5	FHB4	FHB3	FHB2	FHB1	FHB0

读取熔丝位扩展字节时，需要将 0x0002 赋予给 Z 指针并且置位 SPMCSR 寄存器的 BLBSET 和 SPMEN。在SPMCSR操作之后的三个CPU周期内执行的LPM指令将把熔丝位扩展字节的值 (EFB) 加载到目的寄存器。更详细的说明及熔丝位扩展字节映射的细节请参见 P 269Table 117 .

Bit	7	6	5	4	3	2	1	0
Rd	–	–	–	–	–	–	EFB1	EFB0

被编程的熔丝位和锁定位的读返回值为 "0”。未被编程的熔丝位和锁定位的读返回值为 "1”。

防止 Flash 的内容损毁

V_{CC} 低于工作电压时 ,CPU 和 Flash 正常工作无法保证 ,Flash 的内容可能受到破坏。这个问题对于应用于板级系统的独立 Flash 一样存在。所以也要采用同样的解决方案。



电压太低时有两种情况可以破坏 Flash 内容。第一，Flash 写过程需要一个最低电压。第二，电压太低时 CPU 本身会错误地执行指令。

通过遵循以下设计建议可以避免 Flash 被破坏 (采用其中之一就足够了) :

1. 如果系统不需要更新 Boot Loader，建议编程 Boot Loader 锁定位以防止 Boot Loader 软件更新
2. 电源电压不足期间，保持 AVR RESET 为低 : 采用的方式为 : 如果工作电压与检测电平相匹配，可以使能 BOD 功能；否则可以使用外部复位保护电路。如果在写操作进行中发生了复位，只要电源电压足够，写操作还会完成。
3. 低电压期间保持 AVR 内核处于掉电休眠模式。这样可以防止 CPU 解码并执行指令，有效地保护 SPMCSR 寄存器，从而保护 Flash 被无意识得修改掉。

使用 SPM 时的 Flash 编程时间

片内校准的 RC 振荡器用于 Flash 寻址时序控制。 Table 111 给出了 CPU 访问 Flash 的典型编程时间。

Table 111. SPM 编程时间

符号	最小编程时间	最大编程时间
Flash 写操作 (通过 SPM 实现页擦除、页写、及写锁定位)	3.7 ms	4.5 ms

一个简单的引导程序汇编代码

```
  ;- 本例程将 RAM 中的一页数据写入 Flash
  ; Y 指针指向 RAM 的第一个数据单元
  ;Z 指针指向 Flash 的第一个数据单元
  ;- 本例程没有包括错误处理
  ;- 该程序必须放置于 Boot 区 ( 至少 Do_spm 子程序是如此 )
  ; 在自编程过程中 ( 页擦除和页写操作 ) 只能读访问 NRWW 区的代码
  ;- 使用的寄存器 : r0、 r1、 temp1 (r16)、 temp2 (r17)、 looplo (r24)、
  ; loophi (r25)、 SPMCSRval (r20)
  ; 在程序中不包括寄存器内容的保护和恢复
  ; 在牺牲代码大小的情况下可以优化寄存器的使用
  ;- 假设中断向量表位于 Boot loader 区，或者中断被禁止。
.equ  PAGESIZEB = PAGESIZE*2        ;PAGESIZEB 是以字节为单位的页大小，不是以字为单
位
.org SMALLBOOTSTART
Write_page:
  ; 页擦除
  ldi   SPMCSRval, (1<<PGERS) | (1<<SPMEN)
  call  Do_spm

  ; 重新使能 RWW 区
  ldi   SPMCSRval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm

  ; 将数据从 RAM 转移到 Flash 页缓冲区
  ldi   looplo, low(PAGESIZEB)   ; 初始化循环变量
  ldi   loophi, high(PAGESIZEB)  ;PAGESIZEB<=256 时不需要此操作
Wrloop:
  ld    r0, Y+
  ld    r1, Y+
  ldi   SPMCSRval, (1<<SPMEN)
  call  Do_spm
  adiw  ZH:ZL, 2
  sbiw  loophi:looplo, 2        ;PAGESIZEB<=256 时请使用 subi
  brne  Wrloop

  ; 执行页写
```

```
  subi  ZL, low(PAGESIZEB)      ; 复位指针
  sbci  ZH, high(PAGESIZEB)     ;PAGESIZEB<=256 时不需要此操作
  ldi   SPMCSRval, (1<<PGWRT) | (1<<SPMEN)
  call  Do_spm

  ; 重新使能 RWW 区
  ldi   SPMCSRval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm

  ; 读回数据并检查，为可选操作
  ldi   looplo, low(PAGESIZEB)  ; 初始化循环变量
  ldi   loophi, high(PAGESIZEB) ;PAGESIZEB<=256 时不需要此操作
  subi  YL, low(PAGESIZEB)      ; 复位指针
  sbci  YH, high(PAGESIZEB)
Rdloop:
  lpm   r0, Z+
  ld    r1, Y+
  cpse  r0, r1
  jmp   Error
  sbiw  loophi:looplo, 1        ;PAGESIZEB<=256 时请使用 subi
  brne  Rdloop

  ; 返回到 RWW 区
  ; 确保 RWW 区已经可以安全读取
Return:
  lds   temp1, SPMCSR
  sbrs  temp1, RWWSB          ; 若 RWWSB 为 "1"，说明 RWW 区还没有准备好
  ret
  ; 重新使能 RWW 区
  ldi   SPMCSRval, (1<<RWWSRE) | (1<<SPMEN)
  call  Do_spm
  rjmp  Return

Do_spm:
  ; 检查先前的 SPM 操作是否已经完成
Wait_spm:
  lds   temp1, SPMCSR
  sbrc  temp1, SPMEN
  rjmp  Wait_spm
  ;; 输入：SPMCSRval 决定了 SPM 操作
  ; 禁止中断，保存状态标志
  in    temp2, SREG
  cli
  ; 确保没有 EEPROM 写操作
Wait_ee:
  sbic  EECR, EEWE
  rjmp  Wait_ee
  ; SPM 时间序列
  sts   SPMCSR, SPMCSRval
  spm
  ; 恢复 SREG （如果中断原本是使能的，则使能中断）
  out   SREG, temp2
  ret
```

ATmega128 引导程序参数

自编程描述中所用的参数在 Table 112 到 Table 114 中给出。

Table 112. Boot 区大小配置

BOOTSZ1	BOOTSZ0	Boot 区大小	页数	应用 Flash 区	Boot Loader Flash 区	应用区结束地址	Boot 复位地址 (Boot Loader 起始地址)
1	1	512 字	4	$0000 - $FDFF	$FE00 - $FFFF	$FDFF	$FE00
1	0	1024 字	8	$0000 - $FBFF	$FC00 - $FFFF	$FBFF	$FC00
0	1	2048 字	16	$0000 - $F7FF	$F800 - $FFFF	$F7FF	$F800
0	0	4096 字	32	$0000 - $EFFF	$F000 - $FFFF	$EFFF	$F000

Note: 不同的 BOOTSZ 熔丝位配置请参见 Figure 133。

Table 113. RWW 界限 (1)

Flash 区	页数	寻址范围
同时读 - 写区 (RWW)	480	$0000 - $EFFF
非同时读 - 写区 (NRWW)	32	$F000 - $FFFF

Note: 1. 关于两个区的详细说明请见 P 256“ 非 RWW 区 - NRWW” 与 P 255“RWW 区 ” 。

Table 114. Figure 134 中所用变量的说明及 Z 指针的映射 [3]

变量		相应的 Z 指针数据	描述 [2]
PCMSB	15		程序计数器的最高位 (程序计数器为 16 位 PC[15:0])
PAGEMSB	6		用于页内字寻址的最高位 (一页有 128 个字，需要 7 位 PC [6:0]).
ZPCMSB		Z16[1]	Z 寄存器与 PCMSB 对应的位。由于没有使用 Z0 ， ZPCMSB 等于 PCMSB + 1
ZPAGEMSB		Z7	Z 寄存器与 PAGEMSB 对应的位。由于没有使用 Z0 ， ZPAGEMSB 等于 PAGEMSB + 1
PCPAGE	PC[15:7]	Z16[1]:Z7	程序计数器页地址：在页擦除和页写操作中进行页选择
PCWORD	PC[6:0]	Z7:Z1	程序计数器字地址：为填充临时缓冲区进行字选择 (在页写过程中必须为 0)

Notes:
1. Z 寄存器指针为 16 位宽。位 16 位于 RAMPZ 寄存器。
2. Z0：对所有的 SPM 命令都为 "0"，对 (E)LPM 指令则用于字节选择。
3. 关于自编程过程中 Z 指针的使用请参见 P 260“ 在自编程时访问 Flash” 。



存储器编程

程序及数据存储器锁定位

ATmega128 提供了 6 个锁定位，根据其被编程 ("0") 还是没有被编程 ("1") 的情况可以获得 Table116 列出的附加性能。锁定位只能通过芯片擦除命令擦写为 "1"。

Table 115. 锁定位字节

锁定位字节	位号	描述	默认值
	7	–	1 (未编程)
	6	–	1 (未编程)
BLB12	5	Boot 锁定位	1 (未编程)
BLB11	4	Boot 锁定位	1 (未编程)
BLB02	3	Boot 锁定位	1 (未编程)
BLB01	2	Boot 锁定位	1 (未编程)
LB2	1	锁定位	1 (未编程)
LB1	0	锁定位	1 (未编程)

Note: "1" 表示未编程，"0" 表示被编程。

Table 116. 锁定位保护模式

存储器锁定位			保护类型
LB 模式	**LB2**	**LB1**	
1	1	1	没有使能存储器保护特性
2	1	0	在并行和SPI/JTAG串行编程模式中Flash和EEPROM的进一步编程被禁止，熔丝位被锁定。(1)
3	0	0	在并行和SPI/JTAG串行编程模式中Flash和EEPROM的进一步编程及验证被禁止，锁定位和熔丝位被锁定 (1)
BLB0 模式	**BLB02**	**BLB01**	
1	1	1	SPM 和 (E)LPM 对应用区的访问没有限制
2	1	0	不允许 SPM 对应用区进行写操作
3	0	0	不允许 SPM 指令对应用区进行写操作，也不允许运行于 Boot Loader 区的 (E)LPM 指令从应用区读取数据。若中断向量位于 Boot Loader 区，那么执行应用区代码时中断是禁止的。
4	0	1	不允许运行于 Boot Loader 区的 (E)LPM 指令从应用区读取数据。若中断向量位于 Boot Loader 区，那么执行应用区代码时中断是禁止的。
BLB1 模式	**BLB12**	**BLB11**	
1	1	1	允许 SPM/(E)LPM 指令访问 Boot Loader 区

Table 116. 锁定位保护模式

存储器锁定位			保护类型
2	1	0	不允许 SPM 指令对 Boot Loader 区进行写操作
3	0	0	不允许 SPM 指令对 Boot Loader 区进行写操作，也不允许运行于应用区的 (E)LPM 指令从 Boot Loader 区读取数据。若中断向量位于应用区，那么执行 Boot Loader 区代码时中断是禁止的。
4	0	1	不允许运行于应用区的 (E)LPM 指令从 Boot Loader 区读取数据。若中断向量位于应用区，那么执行 Boot Loader 区代码时中断是禁止的。

Notes: 1. 在编程 LB1 和 LB2 前先编程熔丝位和 Boot 锁定位。
2. “1” 表示未被编程 , “0” 表示被编程。

熔丝位

ATmega128 有三个熔丝位字节。Table 117 - Table 119 简单地描述了所有熔丝位的功能以及他们是如何映射到熔丝字节的。如果熔丝位被编程则读返回值为 “0”。

Table 117. 熔丝位扩展字节

扩展熔丝位字节	位号	描述	默认值
–	7	–	1
–	6	–	1
–	5	–	1
–	4	–	1
–	3	–	1
–	2	–	1
M103C(1)	1	ATmega103 兼容模式	0 (编程)
WDTON(2)	0	看门狗定时器始终开启	1 (未编程)

Notes: 1. 详见 P 4“ATmega103 与 ATmega128 的兼容性 ” 。
2. 详见 P 52“ 看门狗定时器控制寄存器 - WDTCR” 。



Table 118. 熔丝位高字节

熔丝位高字节	位号	描述	默认值
OCDEN(4)	7	使能 OCD	1 (未编程， OCD 禁用)
JTAGEN	6	使能 JTAG	0 (编程，JTAG 使能)
SPIEN(1)	5	使能串行程序和数据下载	0 (被编程，SPI 编程使能)
CKOPT(2)	4	振荡器选项	1 (未编程)
EESAVE	3	执行芯片擦除时 EEPROM 的内容保留	1 (未被编程)，EEPROM 内容不保留
BOOTSZ1	2	选择 Boot 区大小 (详见 Table 112)	0 (被编程)(3)
BOOTSZ0	1	选择 Boot 区大小 (详见 Table 112)	0 (被编程)(3)
BOOTRST	0	选择复位向量	1 (未被编程)

Notes: 1. 在 SPI 串行编程模式下 SPIEN 熔丝位不可访问。
2. CKOPT 熔丝位功能由 CKSEL 位设置决定，详见 P 34" 时钟源 "。
3. BOOTSZ1..0 默认值为最大 Boot 大小，详见 P 266Table 112 。
4. 不论锁位与 JTAGEN 熔丝位设置为什么，产品出厂时不对 OCDEN 编程。对 OCDEN 熔丝位编程后会使能系统时钟的某些部分在所有的休眠模式下运行。这会增加功耗。
5. 若JTAG接口未连接，JTAGEN熔丝位应禁用。以避免JTAG接口TDO引脚的静态电流。

Table 119. 熔丝位低位字节

熔丝位低位字节	位号	描述	默认值
BODLEVEL	7	BOD 触发电平	1 (未被编程)
BODEN	6	BOD 使能	1 (未被编程，BOD 禁用)
SUT1	5	选择启动时间	1 (未被编程)(1)
SUT0	4	选择启动时间	0 (被编程)(1)
CKSEL3	3	选择时钟源	0 (被编程)(2)
CKSEL2	2	选择时钟源	0 (被编程)(2)
CKSEL1	1	选择时钟源	0 (被编程)(2)
CKSEL0	0	选择时钟源	1 (未被编程)(2)

Notes: 1. 对于默认时钟源，SUT1..0 的默认值给出最大的起动时间。详细内容见 P 38Table 14 。
2. CKSEL3..0的默认设置导致了片内RC振荡器运行于1 MHz。详细内容见 P 34Table 6 。

熔丝位的状态不受芯片擦除命令的影响。如果锁定位 1(LB1) 被编程则熔丝位被锁定。在编程锁定位前先编程熔丝位。

锁存熔丝位的数据

器件进入编程模式时熔丝位的值被锁存。其间熔丝位的改变不会生效，直到器件退出编程模式。不过这不适用于 EESAVE 熔丝位。它一旦被编程立即起作用。在正常工作模式中器件上电时熔丝位也被锁存。

标识字节

所有的 Atmel 微控制器都具有一个三字节的标识代码用来区分器件型号。这个代码可以通过串行和并行模式读取，也可以在芯片被锁定时读取。这三个字节分别存储于三个独立的地址空间。

ATmega128 标识字节为：

1. $000: $1E (表示由 Atmel 公司生产)。
2. $001: $97 (表示芯片包含 128KB Flash 存储器)。
3. $002: $02 (当 $001 字节的内容为 $97 时表示这是 ATmega128)。

标定字节

ATmega128 保存内部 RC 振荡器的四种标定值。这些字节位于标识地址空间 0x000、0x0001、0x0002与0x0003的高位字节。分别表示1、2、4与8 MHz。在复位期间 ,1 MHz 值被自动写入 OSCCAL 寄存器。如果使用其它频率，标定值要手动写入，详见 P 38 " 振荡器标定寄存器 - OSCCAL" 。



并行编程的参数，引脚映射及命令

这部分描述了如何对 ATmega128 的 Flash 程序存储器、 EEPROM 数据存储器、存储锁定位及熔丝位进行并行编程和校验。除非另有说明，否则脉冲宽度至少为 250 ns。

信号名称

在这一节 ATmega128 的相关引脚以并行编程信号的名称进行引用，如 Figure 135 和 Table 120 所示。表中没有描述的引脚沿用原来的称谓。

XA1/XA0 决定了给 XTAL1 引脚一个正脉冲时所执行的操作。具体编码请见 Table 122。

给 $\overline{WR}$ 或 $\overline{OE}$ 输入脉冲时所加载的命令决定了要执行的操作。具体命令请参见 Table 123。

Figure 135. 并行编程

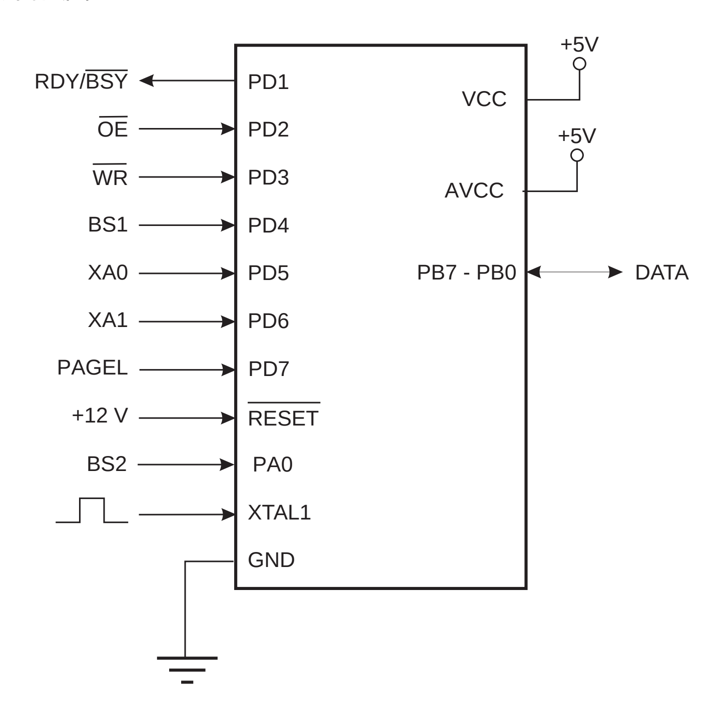


Table 120. 引脚名称映射

编程模式信号的名称	引脚名称	I/O	功能
RDY/$\overline{BSY}$	PD1	O	0: 设备忙于编程，1: 设备等待新的命令。
$\overline{OE}$	PD2	I	输出使能 (低电平有效).
$\overline{WR}$	PD3	I	写脉冲 (低电平有效).
BS1	PD4	I	字节选择 1(“0” 选择低位字节，“1” 选择高位字节).
XA0	PD5	I	XTAL 动作位 0
XA1	PD6	I	XTAL 动作位 1
PAGEL	PD7	I	加载程序存储器和 EEPROM 数据页
BS2	PA0	I	字节选择 2(“0” 选择低位字节，“1” 选择第二个高位字节)
DATA	PB7-0	I/O	双向数据总线 ($\overline{OE}$ 为低时输出)

Table 121. 进入编程模式所需要的引脚数据

引脚	符号	数值
PAGEL	Prog_enable[3]	0
XA1	Prog_enable[2]	0
XA0	Prog_enable[1]	0
BS1	Prog_enable[0]	0

Table 122. XA1 和 XA0 的编码

XA1	XA0	给 XTAL1 施加脉冲激发的操作
0	0	加载 Flash 或 EEPROM 地址 (通过 BS1 确定是高位还是低位字节)
0	1	加载数据 (通过 BS1 决定是高位还是低位 Flash 数据字节)
1	0	加载命令
1	1	无操作，空闲

Table 123. 命令字节编码

命令字节	执行的命令
1000 0000	芯片擦除
0100 0000	写熔丝位
0010 0000	写锁定位
0001 0000	写 Flash
0001 0001	写 EEPROM
0000 1000	读标识字节和校准字节
0000 0100	读熔丝位和锁定位
0000 0010	读 Flash
0000 0011	读 EEPROM

Table 124. 一页包含的字和 Flash 中的页数

Flash 大小	页大小	PCWORD	页号	PCPAGE	PCMSB
64K 字 (128K 字节)	128 字	PC[6:0]	512	PC[15:7]	15

Table 125. 一页包含的字和 EEPROM 中的页数

EEPROM 大小	页大小	PCWORD	页数	PCPAGE	EEAMSB
4K 字节	8 字节	EEA[2:0]	512	EEA[11:3]	8

并行编程

进入并行编程模式

通过下面的算法进入并行编程模式：





1. 在 V_{CC} 及 GND 之间提供 4.5 - 5.5V 的电压，并至少等待 100 μs。
2. 将 $\overline{\text{RESET}}$ 拉低，并至少改变 XTAL1 电平 6 次。
3. 将 P 273Table 121 中列出的的 Prog_enable 引脚置为 "0000"，并等待至少 100 ns。
4. 给 $\overline{\text{RESET}}$ 提供 11.5 - 12.5V 的电压。在向 $\overline{\text{RESET}}$ 提供 +12V 电压后的 100 ns 内，Prog_enable 引脚的任何行为都会导致芯片无法进入编程模式。

如果选择外部晶体或外部 RC，它不可能提供合格的 XTAL1 脉冲。在这种情况下，应采取如下算法：

1. 设置列于 P 273Table 的 Prog_enable 引脚为 “0000”。
2. 在 V_{CC} 与 GND 间提供电压 4.5 - 5.5V 同时在 $\overline{\text{RESET}}$ 上提供 11.5 - 12.5V 电压。
3. 等待 100 ns。
4. 对熔丝位重编程，保证外部时钟源作为系统时钟 (CKSEL3:0 = 0b0000)。如果锁定位已编程，在改变熔丝前必须执行芯片擦除指令。
5. 通过降低器件功率或置 $\overline{\text{RESET}}$ 引脚为 0b0 来退出编程模式。
6. 用前面讲到的算法进入编程模式。

进行高效编程需要考虑的问题

在编程过程中，加载的命令及地址保持不变。为了实现高效的编程应考虑以下因素：

- 对多个存储单元进行读或写操作时，命令仅需加载一次。
- 当需要写入的数据为 $FF 时可以跳过，因为这就是执行全片擦除命令后 Flash 及 EEPROM(除非 EESAVE 熔丝位被编程) 的内容。
- 只有在编程或读取 Flash 及 EEPROM 中新的 256 字时才需要用到地址高位字节。在读标识字节时也需考虑这一点。

芯片擦除

芯片擦除操作会擦除 Flash 及 EEPROM[(1)] 存储器以及锁定位。程序存储器没有擦除结束之前锁定位不会复位。全片擦除不影响熔丝位。芯片擦除命令必须在编程 Flash 与 / 或 EEPROM 之前完成。

Note: 1. 如果 EESAVE 熔丝位被编程，那么在芯片擦除时 EEPRPOM 不受影响。

加载 " 芯片擦除 " 命令的过程：

1. 将 XA1、 XA0 置为 10 以启动命令加载。
2. 将 BS1 置为 0。
3. DATA 赋值为 “1000 0000”。这是芯片擦除命令。
4. 给 XTAL1 提供一个正脉冲，进行命令加载。
5. 给 $\overline{\text{WR}}$ 提供一个负脉冲，启动芯片擦除。 RDY/$\overline{\text{BSY}}$ 变低。
6. 等待 RDY/$\overline{\text{BSY}}$ 变高，然后才能加载新的命令。

对 Flash 进行编程

Flash 是以页的形式组织起来的，如 P 273Table 123 所示。编程 Flash 时，程序数据被锁存到页缓冲区中。这样一整页的程序数据可以同时得到编程。下面的步骤描述了如何对 Flash 进行编程：

A. 加载 " 写 Flash" 命令：

1. 将 XA1、 XA0 置为 "10"，启动命令加载。
2. 将 BS1 置 "0”。
3. DATA 赋值为 “0001 0000”，这是写 Flash 命令。
4. 给 XTAL1 提供一个正脉冲以加载命令。

B. 加载地址低位字节：

1. 将 XA1、 XA0 置为 "00"，启动地址加载。
2. 将 BS1 置 "0”，选择低位地址。

3. DATA 赋值为地址低位字节 ($00 - $FF)。
4. 给 XTAL1 提供一个正脉冲，加载地址低位字节。

C. 加载数据低位字节：

1. 将 XA1、 XA0 置为 "01"，启动数据加载。
2. DATA 赋值为数据低位字节 ($00 - $FF)。
3. 给 XTAL1 提供一个正脉冲，加载数据字节。

D. 加载数据高位字节：

1. 将 BS1 置为 "1”，选择数据高位字节。
2. 将 XA1、 XA0 置为 "01"，启动数据加载。
3. DATA 赋值为数据高位字节 ($00 - $FF)。
4. 给 XTAL1 提供一个正脉冲，进行数据字节加载。

E. 锁存数据：

1. 将 BS1 置为 "1”，选择数据高位字节。
2. 给 PAGEL 提供一个正脉冲，锁存数据 (见 Figure 137 信号波形)。

F. 重复 B 到 E 操作，直到整个缓冲区填满或此页中所有的数据都已加载。

地址信息中的低位用于页内寻址，高位用于 FLASH 页的寻址，详见 P 276Figure 136 。如果页内寻址少于 8 位 (页地址 < 256) , 那么进行页写操作时地址低字节中的高位用于页寻址。

G. 加载地址高位字节：

1. 将 XA1、 XA0 置为 "00”，启动地址加载操作。
2. 将 BS1 置为 "1”，选择高位地址。
3. DATA 赋值为地址高位字节 ($00 - $FF)。
4. 给 XTAL1 提供一个正脉冲，加载地址高位字节。

H. 编程一页数据：

1. 置 BS1 = “0”。
2. 给 $\overline{\text{WR}}$ 提供一个负脉冲，对整页数据进行编程， RDY/$\overline{\text{BSY}}$ 变低。
3. 等待 RDY/$\overline{\text{BSY}}$ 变高 (见 Figure 137 信号波形)。

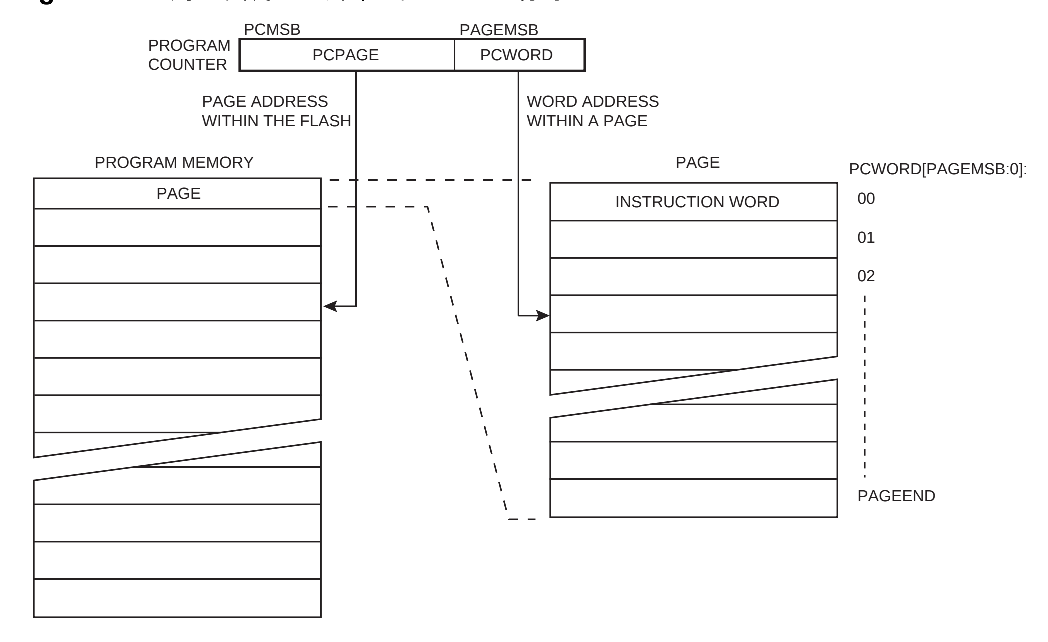
I. 重复 B 到 H 的操作，直到整个 Flash 编程结束或者所有的数据都被编程。

J. 结束页编程：

1. 1. 将 XA1、 XA0 置为 "10”，启动命令加载操作。
2. DATA 赋值为 "0000 0000"，这是不操作指令。
3. 给 XTAL1 提供一个正脉冲，加载命令，内部写信号复位。

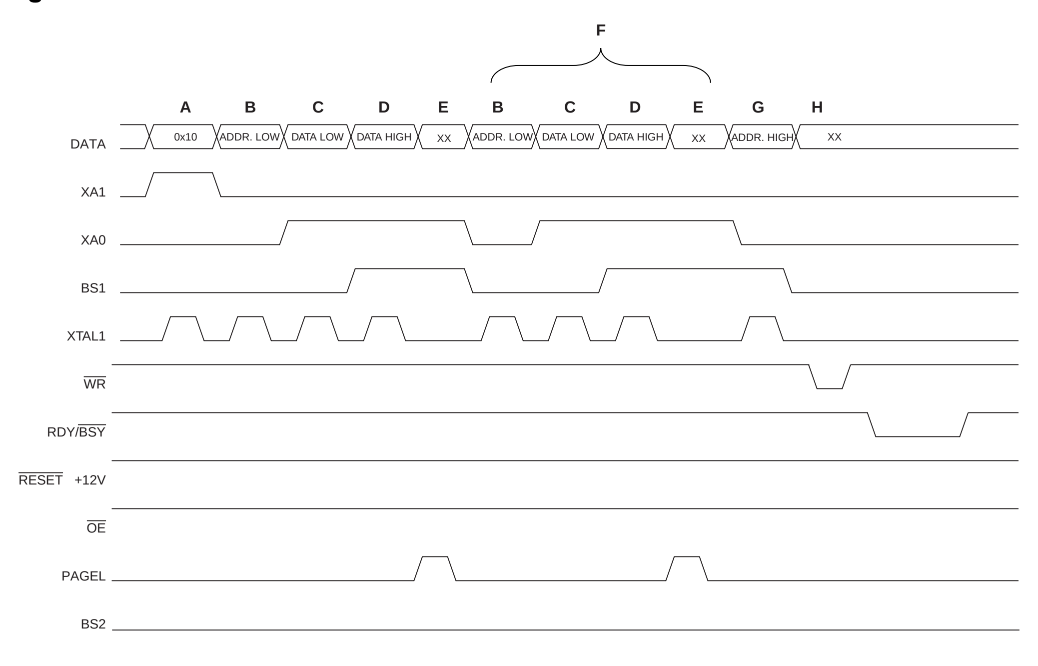


Figure 136. 对以页为组织单位的 Flash 进行寻址



Note: 1. PCPAGE 及 PCWORD 列于 P 273Table 124 。

Figure 137. Flash 编程波形



Note: 不用考虑 "XX"，各个大写字母对应于前面描述的 Flash 编程阶段。

对 EEPROM 进行编程

如 P 273Table 124 所示， EEPROM 也以页为单位。编程 EEPROM 时，编程数据锁存于页缓冲区中。这样可以同时对一页数据进行编程。 EEPROM 数据存储器编程算法如下 (命令、地址及数据加载的细节请参见 P 273“ 对 Flash 进行编程 ”) :

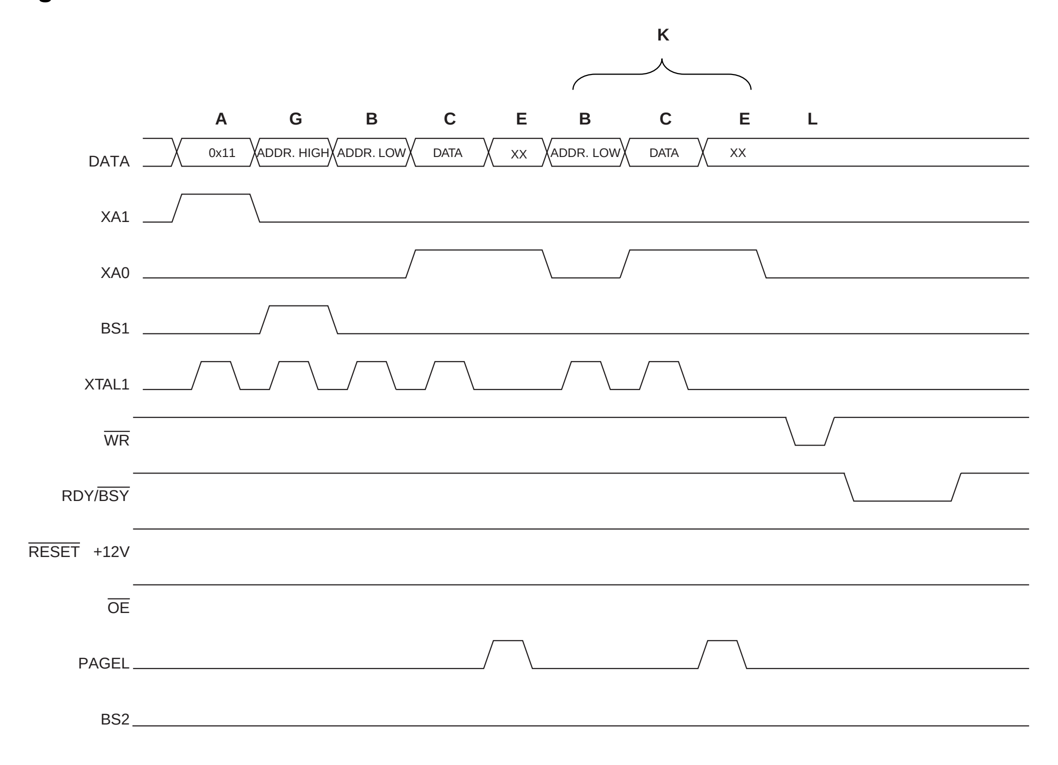
1. A：加载命令 “0001 0001”。
2. G：加载地址高位字节 ($00 - $FF)。
3. B：加载地址低位字节 ($00 - $FF)。

4. C：加载数据 ($00 - $FF)。
5. E：锁存数据 (给 PAGEL 提供一个正脉冲)。

K：重复步骤 3 到 5，直到整个缓冲区填满。

L：对 EEPROM 页进行编程：
1. 将 BS1 置 “0”。
2. 给 $\overline{WR}$ 提供一个负脉冲，开始对 EEPROM 页进行编程， RDY/$\overline{BSY}$ 变低。
3. 等到 RDY/$\overline{BSY}$ 变高再对下一页进行编程 (信号波形见 Figure 138)。

Figure 138. EEPROM 编程波形



读取 Flash

读 Flash 存储器的过程如下 (命令及地址加载细节见 P 273“ 对 Flash 进行编程 ”)：
1. A：加载命令 “0000 0010”。
2. G：加载地址高位字节 ($00 - $FF)。
3. B：加载地址低位字节 ($00 - $FF)。
4. 将 $\overline{OE}$ 置 “0”， BS1 置 “0”，然后从 DATA 读出 Flash 字的低位字节。
5. 将 BS1 置 “1”，然后从 DATA 读出 Flash 字的高位字节。
6. 将 $\overline{OE}$ 置 “1”。

读取 EEPROM

读存储器的步骤如下 (命令及地址加载细节见 P 273“ 对 Flash 进行编程 ”)：
1. A：加载命令 “0000 0011”。
2. G：加载地址高位字节 ($00 - $FF)。
3. B：加载地址低位字节 ($00 - $FF)。
4. 将 $\overline{OE}$ 置 "0”， BS1 置 "0”，然后从 DATA 读出 EEPROM 数据字节。
5. 将 $\overline{OE}$ 置 ”1”。

对熔丝位的低位进行编程

对熔丝低位的编程步骤如下 (命令及数据装在细节见 P 273“ 对 Flash 进行编程 ”)：



1. A：加载命令 “0100 0000”。
2. C：加载数据低字节，若某一位为 "0” 表示需要进行编程，否则需要擦除。
3. 置 BS1 为 “0”， BS2 为 “0”。
4. 给 $\overline{WR}$ 提供一个负脉冲，并等待 RDY/$\overline{BSY}$ 变高。

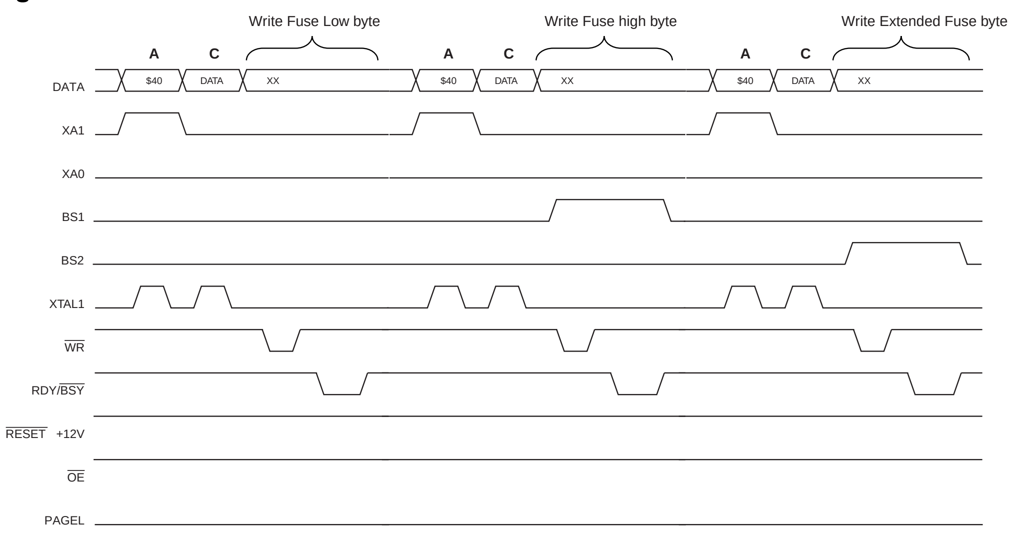
对熔丝位的高位进行编程

对熔丝高位的编程步骤如下 (命令及数据装在细节见 P 273“ 对 Flash 进行编程 ”) :
1. A：加载命令 “0100 0000”。
2. C：加载数据高字节，若某一位为 "0” 表示需要进行编程，否则需要擦除。
3. 将 BS1 置 "1”、 BS2 置 "0”，选择高位数据字节。
4. 给 $\overline{WR}$ 提供一个负脉冲并等待 RDY/$\overline{BSY}$ 变高。
5. 将 BS1 置 "0”，选择低位字节。

对扩展熔丝位进行编程

对扩展熔丝位的编程步骤如下 (命令及数据装在细节见 P 273“ 对 Flash 进行编程 ”) :
1. A：加载命令 “0100 0000”。
2. C：加载数据低字节。若某一位为 "0” 表示需要进行编程，否则需要擦除。
3. 将 BS1 置 "0”、 BS2 置 "1”，选择扩展数据字节。
4. 给 $\overline{WR}$ 提供一个负脉冲并等待 RDY/$\overline{BSY}$ 变高。
5. 将 BS2 置 "0”，选择低位字节。

Figure 139. 熔丝位编程波形



对锁定位进行编程

锁定位编程步骤如下 (命令及数据装在细节见 P 273“ 对 Flash 进行编程 ”) :
1. A：加载命令 “0010 0000”。
2. C.：加载数据低字节，位 n 为 "0” 表示此锁定位需要编程。
3. 给 $\overline{WR}$ 提供一个负脉冲并等待 RDY/$\overline{BSY}$ 变高。

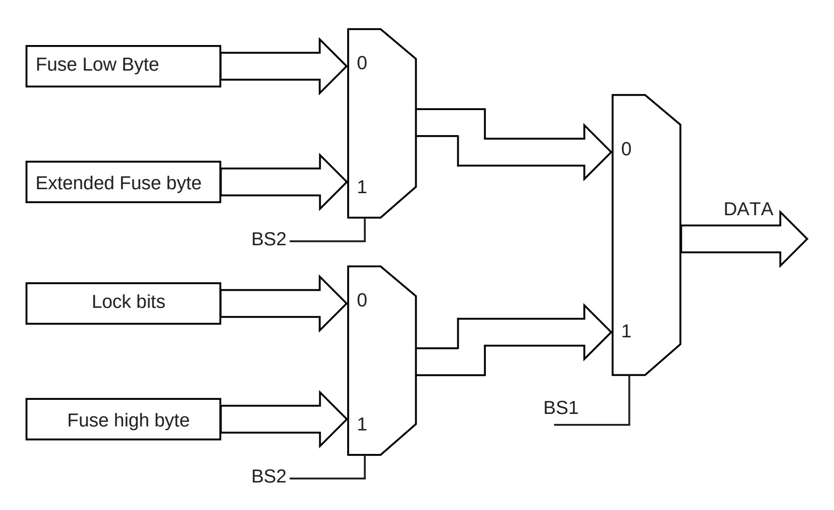
锁定位只能通过芯片擦除命令来清除。

读取熔丝位和锁定位

读取熔丝位及锁定位的步骤如下 (命令加载细节见 P 273“ 对 Flash 进行编程 ”) :
1. A：加载命令 “0000 0100”。
2. 将 $\overline{OE}$、BS2 和 BS1 置 "0” , 然后从 DATA 读取熔丝低位的状态 ("0” 表示已编程)。
3. 将$\overline{OE}$置"0” ,BS2和BS1置“1” , 然后从DATA读取熔丝高位的状态("0”表示已编程)。

4. 将 $\overline{OE}$ 和BS1置"0” ,BS2置“1” ,然后从DATA读取扩展熔丝位的状态("0”表示已编程)。
5. 将 $\overline{OE}$ 置 "0” ,BS2 置 "0” ,BS1 置 “1” ,然后从 DATA 读取锁定位的状态 ("0” 表示已编程)。
6. 将 $\overline{OE}$ 置 “1”。

Figure 140. 读操作过程中 BS1、 BS2 与熔丝位及锁定位的对应关系



读取标识字节

读取标识字节的算法如下 (命令与地址加载参考对 Flash 编程) :

1. A : 加载命令 “0000 1000”。
2. B : 加载地址低字节 ($00 - $02)。
3. 将 $\overline{OE}$ 、 BS1 置 "0” , 然后从 DATA 读取标识字节。
4. 将 $\overline{OE}$ 置 "1”。

读取标定字节

读取校准字节的算法如下 (命令与地址加载参考对 Flash 编程) :

1. A : 加载命令 “0000 1000”。
2. B : 加载地址低字节。
3. 将 $\overline{OE}$ 置 "0” , BS1 置 “1” , 然后从 DATA 读取校准字节。
4. 将 $\overline{OE}$ 置 “1”。

并行编程特性

Figure 141. 并行编程时序 , 包括一些常规的时序要求

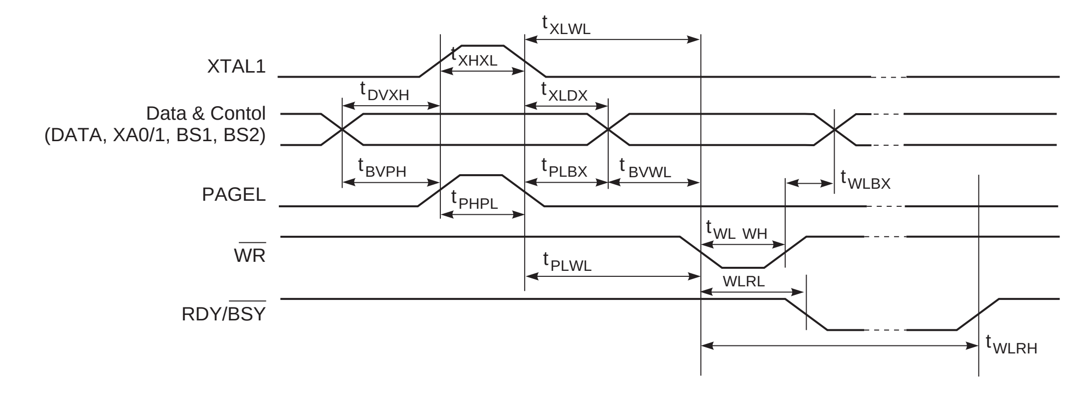
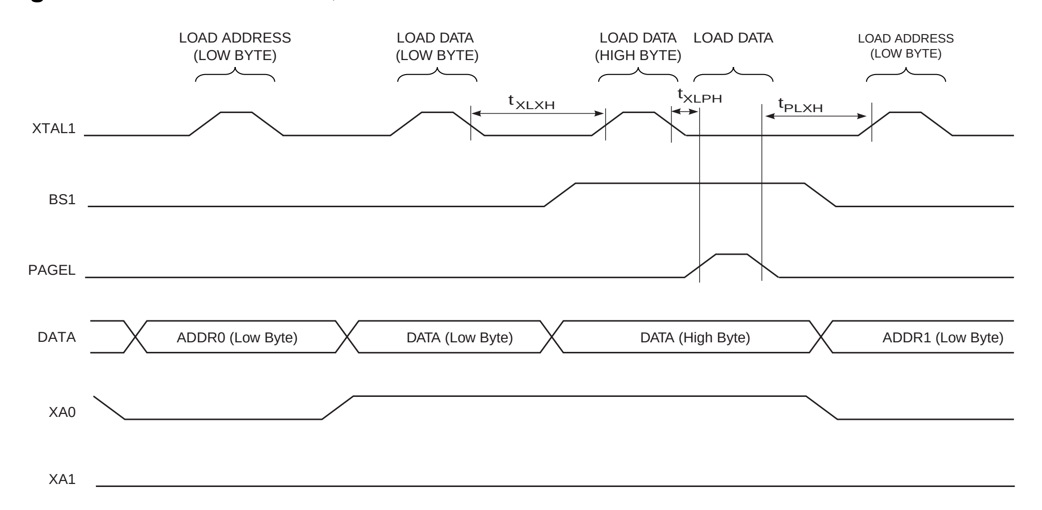
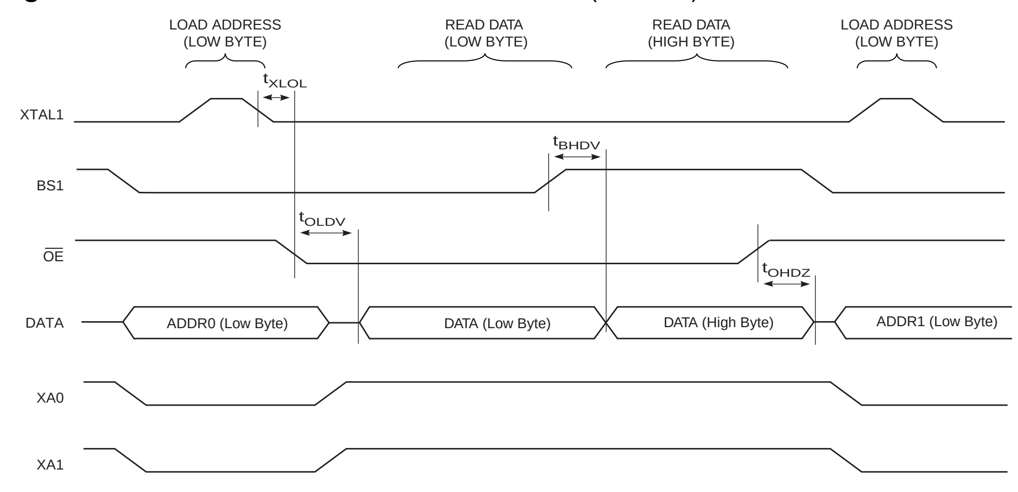


Figure 142. 并行编程时序，有时序要求的加载序列



Note: Figure 141 给出的时序要求 (t_{DVXH}、 t_{XHXL} 及 t_{XLDX}) 也适用于加载操作

Figure 143. 并行编程时序，有时序要求的读序列 (同一页)



Note: Figure 141 给出的时序要求 (即 t_{DVXH}、 t_{XHXL} 及 t_{XLDX}) 也适用于读操作。

Table 126. 并行编程参数 V_{CC} = 5 V ± 10%

符号	参数	最小值	典型值	最大值	单位
V_{PP}	编程使能电压	11.5		12.5	V
I_{PP}	编程使能电流			250	μA
t_{DVXH}	在 XTAL1 为高之前数据及控制有效	67			ns
t_{XLXH}	从 XTAL1 低到 XTAL1 高	200			ns
t_{XHXL}	XTAL1 为高时的脉宽	150			ns
t_{XLDX}	XTAL1 为低之后数据及控制保持	67			ns
t_{XLWL}	从 XTAL1 低到 $\overline{WR}$ 低	0			ns
t_{XLPH}	从 XTAL1 低到 PAGEL 高	0			ns
t_{PLXH}	从 PAGEL 低到 XTAL1 高	150			ns

Table 126. 并行编程参数 V_{CC} = 5 V ± 10%

符号	参数	最小值	典型值	最大值	单位
t_{BVPH}	PAGEL 为高之前 BS1 有效	67			ns
t_{PHPL}	PAGEL 为高时的脉宽	150			ns
t_{PLBX}	PAGEL 为低之后 BS1 保持	67			ns
t_{WLBX}	$\overline{WR}$ 为低之后 BS2/1 保持	67			ns
t_{PLWL}	从 PAGEL 低到 $\overline{WR}$ 为低	67			ns
t_{BVWL}	BS1 有效至 $\overline{WR}$ 为低	67			ns
t_{WLWH}	$\overline{WR}$ 为低时的脉宽	150			ns
t_{WLRL}	从 $\overline{WR}$ 低到 RDY/$\overline{BSY}$ 为低	0		1	µs
t_{WLRH}	从 $\overline{WR}$ 低到 RDY/$\overline{BSY}$ 为高 (1)	3.7		4.5	ms
t_{WLRH_CE}	从 $\overline{WR}$ 低到 RDY/$\overline{BSY}$ 为高，芯片擦除操作 (2)	7.5		9	ms
t_{XLOL}	从 XTAL1 低到 $\overline{OE}$ 为低	0			ns
t_{BVDV}	BS1 有效至 DATA 有效	0		250	ns
t_{OLDV}	从 $\overline{OE}$ 低到 DATA 有效			250	ns
t_{OHDZ}	从 $\overline{OE}$ 低到 DATA 为三态			250	ns

Notes: 1. 在进行 Flash、 EEPROM、熔丝位及锁定位写操作时 t_{WLRH} 有效。
2. 在执行芯片擦除操作时 t_{WLRH_CE} 有效。

串行下载

当 $\overline{RESET}$ 位低电平时，可以通过串行 SPI 总线对 Flash 及 EEPROM 进行编程。串行接口包括 SCK、 MOSI(输入) 及 MISO(输出)。 $\overline{RESET}$ 为低之后，应在执行编程 / 擦除操作之前执行编程允许指令。P 281Table 127 列出了 SPI 编程所需引脚的映射。不是所有的器件都使用 SPI 引脚专用于内部 SPI 接口。注意，在串行载入的说明中， MOSI 与 MISO分别描述连续数据的输入与输出。在ATmega128中，这些引脚映射为PDI 与PDO。

SPI 串行编程引脚映射

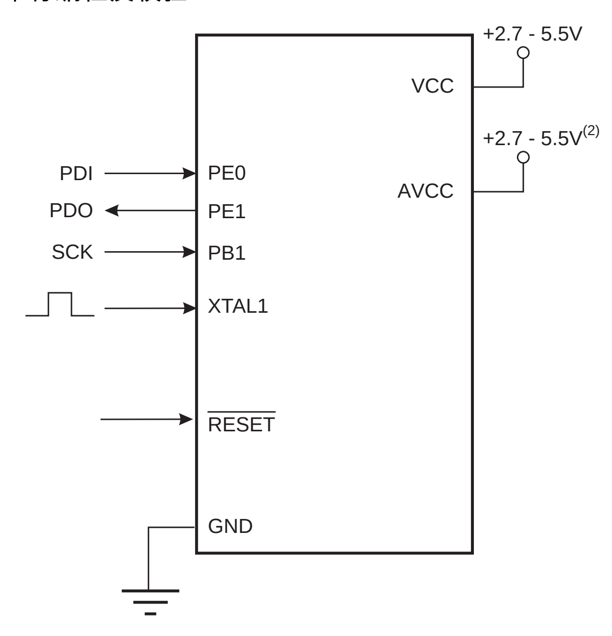
尽管 SPI 编程接口重用 SPI I/O 模块，但其中有一点不同：MOSI/MISO 引脚映射到 SPI I/O模块的PB2与PB3在编程接口并不使用。而PE0与PE1用来传递数据，如Table 127所示。

Table 127. SPI 串行编程映射

符号	引脚	I/O	说明
MOSI (PDI)	PE0	I	连续数据输入
MISO (PDO)	PE1	O	连续数据输出
SCK	PB1	I	连续时钟



Figure 144. SPI 串行编程及校验 (1)



Notes: 1. 如果芯片由片内振荡器提供时钟，那么就不用在 XTAL1 引脚上连接时钟源。
2. V_{CC} - 0.3V < AVCC < V_{CC} + 0.3V，但是 AVCC 必须在 2.7 - 5.5V 范围内。

编程 EEPROM 时， MCU 在自定时的编程操作中会插入一个自动擦除周期，从而无需执行芯片擦除命令。芯片擦除操作将程序存储器及 EEPROM 的内容都擦除为 $FF。

时钟通过 CKSEL 熔丝位确定。串行时钟 (SCK) 的最小低电平时间和最小高电平时间要满足如下要求：

低 :> f_{ck} < 12 MHz 时为 2 个 CPU 时钟周期，f_{ck} >= 12 MHz 时为 3 个 CPU 时钟周期。

高 :> f_{ck} < 12 MHz 时为 2 个 CPU 时钟周期，f_{ck} >= 12 MHz 时为 3 个 CPU 时钟周期。

SPI 串行编程算法

向 ATmega128 串行写入数据时，数据在 SCK 的上升沿得以锁存。

从 ATmega128 读取数据时，数据在 SCK 的下降沿输出。时序细节见 Figure 145。

在串行编程模式下对 ATmega128 进行编程及校验时，应遵循以下的步骤 (见 Table 145 中的 4 字节指令格式)：

1. 上电顺序：
   在 $\overline{RESET}$ 及 SCK 为 "0” 时，向 V_{CC} 及 GND 供电。在一些系统中，编程器不能保证在上电时 SCK 保持为低。在这种情况下， SCK 拉低之后应在 $\overline{RESET}$ 加一正脉冲，而且这个脉冲至少要维持 2 个 CPU 时钟周期。
   或者也可在上电复位时 $\overline{PEN}$ 为低且将 SCK 置为 “0”，代替 RESET 信号功能。在这种情况下，上电复位时的 $\overline{PEN}$ 值最重要。如果程序员不能保证上电过程中 SCK 为低，则不能使用这种方法。使用此方法时，器件必须掉电，以便开始正常操作。
2. 上电之后等待至少 20 ms，然后向 MOSI 引脚输入串行编程使能指令以使能串行编程。
3. 通信不同步将造成串行编程指令不工作。同步之后，在发送编程使能指令的第三个字节时，第二个字节的内容 ($53) 将被反馈回来。不论反馈的内容正确与否，指令的 4 个字节必须全部传输。如果 $53 未被反馈，则需要向 $\overline{RESET}$ 提供一个正脉冲以开始新的编程使能指令。

4. Flash 的编程以一次一页的方式进行。页尺寸见 P 273Table 124 。在执行加载程序存储页指令时，通过 7 LSB 的地址信息，数据以字节为单位加载到存储页。为保证加载的正确性，应先向给定地址传送数据低字节，之后是高字节。程序存储页通过地址的高 9 位以及写程序存储器页指令获得数据。如果不使用查询的方式，那么在操作下一页数据之前应等待至少 t_{WD_FLASH} 的时间 (见 Table 128)。在 Flash 写操作完成之前访问串行编程接口会导致编程错误。

   Note: 若其他命令 (除轮询读) 在写操作完成前出现 (Flash、EEPROM、锁定位、熔丝位)，可能会导致编程出错。
5. 提供了地址及数据信息之后，适合的写指令将以字节为单位对 EEPROM 编程。EEPROM 存储单元总是在写入新数据之前自动擦除。如果不使用查询的方式，那么在操作下一页数据之前应等待至少 t_{WD_EEPROM} 的时间 (见 Table 128) 。对于全片擦除之后的芯片，数据为 0xFF 的不需要编程。
6. 可通过读指令来校验任何一个存储单元的内容。数据从串行输出口 MISO 输出。
7. 编程结束后可以将 $\overline{\text{RESET}}$ 拉高开始正常操作。
8. 下电序列 (如果需要)：
   将 $\overline{\text{RESET}}$ 置 “1”。
   切断 V_{CC}。

Flash 的数据轮询

当 Flash 正处于某一页的编程状态时，读取此页中的内容将得到 $FF。编程结束后，被编程的数据即可以正确读出。通过这种方法可以确定何时可以写下一页。由于整个页是同时编程的，这一页中的任何一个地址都可以用来查询。 Flash 数据查询不适用于数据 $FF。因此，在编程 $FF 时，用户至少要等待 t_{WD_FLASH} 才能进行下一页的编程。由于全片擦除将所有的单元擦为 $FF，所以编程数据为 0xFF 时可以跳过这个操作。 t_{WD_FLASH} 的值见 Table 128。

EEPROM 的数据轮询

当 EEPROM 正在处理一个字节的编程操作时，读取此地址将返回 $FF。编程结束后，被编程的数据即可以正确读出。这一方法可用来判断何时可以写下一个字节。数据查询对数据 $FF 无效。但用户应该考虑到，全片擦除将所有的单元擦为 $FF ,所以编程数据为 $FF 时可以跳过这个操作。不过这不适用于全片擦除时 EEPROM 内容被保留的情况。用户若在此时编程 $FF，在进行下一字节编程之前至少等待 t_{WD_EEPROM} 的时间。t_{WD_EEPROM} 的值见 Table 128。

Table 128. 写下一个 Flash 或 EEPROM 单元之前的最小等待时间 , V_{CC} = 5 V ± 10%

符号	最小等待时间
t_{WD_FUSE}	5 ms
t_{WD_FLASH}	5 ms
t_{WD_EEPROM}	10 ms
t_{WD_ERASE}	10 ms

Figure 145. .SPI 串行编程波形图

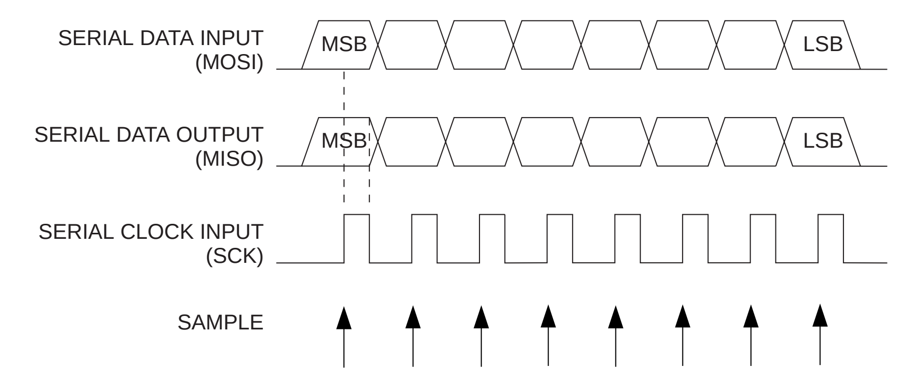


Table 129. SPI 串行编程指令集

指令	指令格式				操作
	字节 1	字节 2	字节 3	字节 4	
编程使能	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	$\overline{\text{RESET}}$ 拉低后使能串行编程
全片擦除	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	擦除 EEPROM 及 Flash
读程序存储器	0010 **H**000	**aaaa aaaa**	**bbbb bbbb**	**oooo oooo**	从字地址为 **a**:**b** 的程序存储器读取 **H**(高或低字节) 数据的 **o**
加载程序存储器页	0100 **H**000	xxxx xxxx	x**bbb bbbb**	**iiii iiii**	向字地址为 **b** 的程序存储页 **H**(高或低字节) 写入数据 **i**。应先写低字节再写高字节
写程序存储器页	0100 1100	**aaaa aaaa**	**b**xxx xxxx	xxxx xxxx	在地址 **a**:**b** 加载程序存储页
读 EEPROM 存储器	1010 0000	xxxx **aaaa**	**bbbb bbbb**	**oooo oooo**	从 EEPROM 的地址 **a**:**b** 处读出数据 **o**
写 EEPROM 存储器	1100 0000	xxxx **aaaa**	**bbbb bbbb**	**iiii iiii**	向 EEPROM 地址 **a**:**b** 处中写入数据 **o**
读锁定位	0101 1000	0000 0000	xxxx xxxx	xx**oo oooo**	读锁定位。 ”0” 为已编程， "1” 为未编程。细节见 P 268Table 115 。
写锁定位	1010 1100	111x xxxx	xxxx xxxx	11**ii iiii**	写锁定位。写 “0” 表示编程锁定位。细节见 P 268Table 115 。
读标识字节	0011 0000	xxxx xxxx	xxxx xx**bb**	**oooo oooo**	从地址 **b** 读取标识字节 **o**
写熔丝位	1010 1100	1010 0000	xxxx xxxx	**iiii iiii**	“0” 表示已编程， “1” 表示未编程。见 P 270Table 119
写高熔丝位	1010 1100	1010 1000	xxxx xxxx	**iiii iiii**	“0” 表示已编程， “1” 表示未编程。见 P 270Table 118
写扩展熔丝位	1010 1100	1010 0100	xxxx xxxx	xxxx xx**ii**	“0” 表示已编程， “1” 表示未编程。见 P 270Table 119
读熔丝位	0101 0000	0000 0000	xxxx xxxx	**oooo oooo**	“0” 表示已编程， “1” 表示未编程。见 P 270Table 119
读扩展熔丝位	0101 0000	0000 1000	xxxx xxxx	**oooo oooo**	读扩展熔丝位。“0” 表示已编程， “1” 表示未编程。细节见 P 270Table 119
读高熔丝位	0101 1000	0000 1000	xxxx xxxx	**oooo oooo**	读熔丝高位。 “0” 表示已编程， “1” 表示未编程。细节见 P 270Table 118
读校准字节	0011 1000	xxxx xxxx	0000 00**bb**	**oooo oooo**	在地址 **b** 读校准字节 **o**

Note: **a** = 地址高位
b = 地址低位
H = 0 - 低字节， 1 - 高字节
o = 数据输出 t
i = 数据输入
x = 任意值

SPI 串行编程特性

SPI 模块特性，见 P 303“SPI 时序特性 ” 。

通过 JTAG 接口进行编程

通过 JTAG 接口进行编程需要控制 4 个 JTAG 专用引脚 :TCK、TMS、TDI 及 TDO。reset 及时钟引脚不用控制。

使用 JTAG 接口之前首先要编程 JTAGEN 熔丝位。芯片出厂时这个熔丝位缺省为编程状态。此外 , MCUCSR 寄存器的 JTD 位必须清零。如果 JTD 已被置 1 , 则可以将外部 reset 强制拉低。经过两个时钟周期之后 JTD 位就清零了。 JTAG 引脚即可用于编程功能。因此，JTAG 引脚除了可以用作通用 I/O 之外还可以用于 ISP 功能。但要注意，当 JTAG 用于边界扫描或片内调试时，不可以使用这个技术。在这种情况下，JTAG 引脚只能用作上述用途。

LSB 是第一个移入 / 移出移位寄存器的位。

与编程相关的 JTAG 指令

指令寄存器为 4 位，可支持 16 种指令。用于编程的 JTAG 指令在下面列出。

每一条指令的执行代码 OPCODE 都在指令名后以 16 进制形式列出。文字则描述了每条指令中 TDI 和 TDO 之间以哪个被数据寄存器作为数据通路。

TAP 控制器的 Run-Test/Idle 状态用来产生内部时钟。它也可用作 JTAG 序列之间的空闲状态。改变指令字的状态机序列见 Figure 146。

Figure 146. 改变指令字的状态机序列

AVR_RESET ($C)	AVR_RESET为AVR特有的JTAG指令，用于使AVR进入复位模式或退出复位模式。这条指令不能进行 TAP 的重置。字长只有 1 位的复位寄存器被用作数据寄存器。一旦复位链中有一个逻辑 1，复位状态立刻被激活。这条链的输出不锁存。

活动状态是：

- Shift-DR：复位寄存器通过输入的 TCK 时钟进行移位。

PROG_ENABLE ($4)

PROG_ENABLE是AVR专用的JTAG指令，用来使能JTAG编程。16位的编程使能寄存器被选作数据寄存器。活动状态为：

- Shift-DR：编程使能标志被移入数据寄存器。
- Update-DR：编程使能标志与正确的数值进行比较，如果标志有效即进入编程模式。

PROG_COMMANDS ($5)

AVR 专用的 JTAG 指令。用于通过 JTAG 接口输入编程命令。15 位的编程命令寄存器被用作数据寄存器。活动状态有：

- Capture-DR：上一条指令的结果加载到数据寄存器。

- Shift-DR : 数据寄存器的内容通过输入的 TCK 时钟进行移位，将上一条指令的结果移出数据寄存器，并移入新的命令。
- Update-DR：编程命令已应用到 Flash 输入。
- Run-Test/Idle：产生一个时钟来执行加载的命令 (不总是需要，见 Table 130)。

PROG_PAGELOAD ($6)

AVR专用的JTAG指令。其功能是通过JTAG接口直接将数据加载到Flash数据页。2048位的数据字节寄存器被用作数据寄存器。这是长度等于一页 Flash 的虚拟扫描链。内部移位寄存器为 8 位。与大多数 JTAG 指令不同， Update-DR 状态不是用来从移位寄存器中传送数据。在 Shift-DR 状态中，通过内部状态工具，数据自动以字节为单位传入 Flash 页缓冲器，活动状态为：

- Shift-DR：Flash 数据字节寄存器的内容被通过输入的 TCK 时钟进行移位。且每次自动载入一个字节。

Note: JTAG 指令 PROG_PAGELOAD 只有在 AVR 器件是 JTAG 扫描链的的一个器件中使用。若 AVR 不是扫描链上第一个器件，则必须使用字节法则编程算法。

PROG_PAGEREAD ($7)

AVR专用的JTAG指令。其功能是通过JTAG接口读取Flash的内容。2056位虚拟 Flash 页读寄存器作为数据寄存器。这是长度等于一页 Flash 加 8 的虚拟扫描链。内部移位寄存器为 8 位。与大多数 JTAG 指令不同，Capture-DR 状态不是用来传送数据到移位寄存器 。在 Shift-DR 状态中，通过内部状态工具，数据自动以字节为单位从 Flash 页缓冲器传出，活动状态为：

- Shift-DR：Flash 数据字节寄存器的内容通过输入的 TCK 时钟进行移位操作。TDI 输入忽略。

Note: JTAG 的 PROG_PAGEREAD 指令只有当 AVR 器件是 JTAG 扫描链的第一个器件时才能使用。如果器件不满足这一条件，则必须使用字节法则编程算法。

数据寄存器

数据寄存器由 JTAG 指令寄存器选取，如 P 286“ 与编程相关的 JTAG 指令 ” 所述。与编程操作相关的数据寄存器为：

- 复位寄存器
- 编程使能寄存器
- 编程命令寄存器
- 虚拟 Flash 页载入寄存器
- 虚拟 Flash 页读出寄存器

复位寄存器

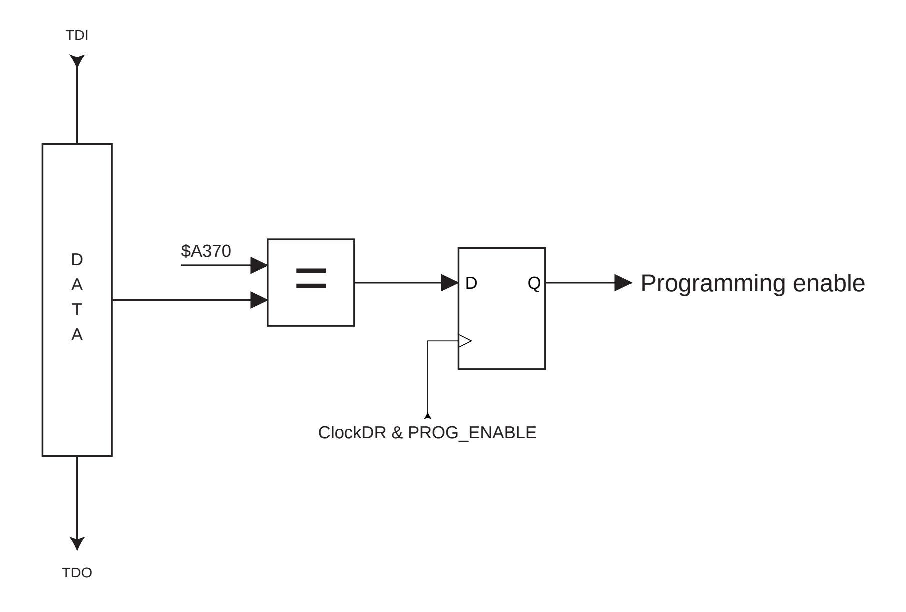
复位寄存器是一个测试数据寄存器，用来在编程期间复位芯片。进入编程模式之前首先要复位器件。

复位寄存器的值大于 0 相当于将外部复位引脚拉低。只要复位寄存器的值大于 0 芯片就处于复位状态。根据时钟项熔丝位的设置，在释放复位寄存器之后，设备仍保持复位状态直到复位定时时间溢出 (见 P 34“ 时钟源 ”) 。从数据寄存器输出的数据不锁存，因此复位会立即发生，如 P 238Figure 123 。

编程使能寄存器

编程使能寄存器是一个 16 位的寄存器。这个寄存器的内容将与编程使能信号 (二进制 1010_0011_0111_0000) 进行比较。如果寄存器的内容与编程使能信号相同，就可以通过 JTAG 进行编程。此寄存器在上电复位时复位，并且在退出编程模式时也应将它复位。

Figure 147. 编程使能寄存器



编程命令寄存器

编程命令寄存器是一个 15 位的寄存器。这个寄存器用来串行地移入编程命令，串行地移出上一个命令的执行结果。 JTAG 编程指令集见 Table 130。编程命令移入的状态序列请参见 Figure 149。

Figure 148. 编程命令寄存器

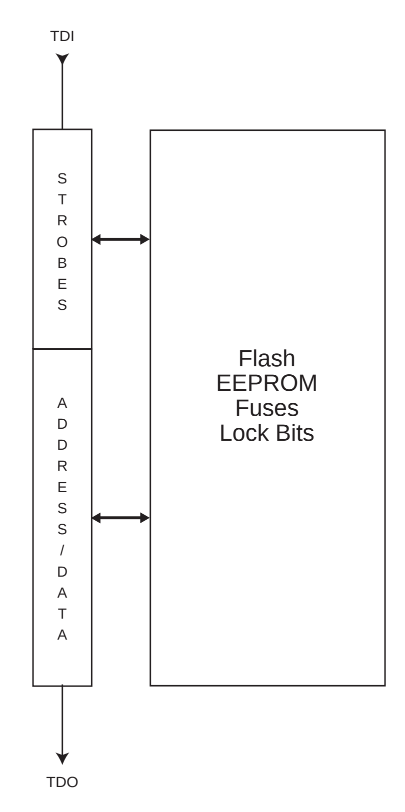


Table 130. JTAG 编程指令集
a = 高位地址 , **b** = 低位地址 , **H** = 0 - 高字节 , 1 - 低字节 , **o** = 数据输出 , **i** = 数据输入 , x = 无用途

指令	TDI 序列	TDO 序列	注意
1a. 芯片擦除	0100011_10000000 0110001_10000000 0110011_10000000 0110011_10000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	
1b. 芯片擦除结束检测	0110011_10000000	xxxxx**o**x_xxxxxxxx	(2)
2a. 进入 Flash 写操作	0100011_00010000	xxxxxxx_xxxxxxxx	
2b. 加载高位地址	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
2c. 加载低位地址	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
2d. 加载数据低字节	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	
2e. 加载数据高字节	0010111_**iiiiiiii**	xxxxxxx_xxxxxxxx	
2f. 锁存数据	0110111_00000000 1110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
2g. 写 Flash 页	0110111_00000000 0110101_00000000 0110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
2h. 检测页写是否结束	0110111_00000000	xxxxx**o**x_xxxxxxxx	(2)
3a. 进入 Flash 读操作	0100011_00000010	xxxxxxx_xxxxxxxx	
3b. 加载高位地址	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
3c. 装载低位地址	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
3d. 读数据低、高字节	0110010_00000000 0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo** xxxxxxx_**oooooooo**	 低字节 高字节
4a. 进入 EEPROM 写操作	0100011_00010001	xxxxxxx_xxxxxxxx	
4b. 加载高位地址	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)
4c. 加载低位地址	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
4d. 加载数据字节	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	
4e. 数据锁存	0110111_00000000 1110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
4f. 写 EEPROM 页	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
4g. 检测页写是否结束	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
5a. 进入 EEPROM 读操作	0100011_00000011	xxxxxxx_xxxxxxxx	
5b. 加载高位地址	0000111_**aaaaaaaa**	xxxxxxx_xxxxxxxx	(9)



Table 130. JTAG 编程指令集
a = 高位地址 , **b** = 低位地址 , **H** = 0 - 高字节 , 1 - 低字节 , **o** = 数据输出 , **i** = 数据输入 , x = 无用途

指令	TDI 序列	TDO 序列	注意
5c. 加载低位地址	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
5d. 读数据字节	0110011_**bbbbbbbb** 0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
6a. 进入写熔丝位操作	0100011_01000000	xxxxxxx_xxxxxxxx	
6b. 加载数据低字节 (6)	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	(3)
6c. 写扩展熔丝位字节	0111011_00000000 0111001_00000000 0111011_00000000 0111011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6d. 检测写熔丝位是否结束	0110111_00000000	xxxxx**o**x_xxxxxxxx	(2)
6e. 加载数据低字节 (7)	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	(3)
6f. 写熔丝位高字节	0110111_00000000 0110101_00000000 0110111_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6g. 检测写熔丝位是否结束	0110111_00000000	xxxxx**o**x_xxxxxxxx	(2)
6h. 加载数据低位字节 (7)	0010011_**iiiiiiii**	xxxxxxx_xxxxxxxx	(3)
6i. 写熔丝位低位字节	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
6j. 检测熔丝位写完	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
7a. 进入写锁定位操作	0100011_00100000	xxxxxxx_xxxxxxxx	
7b. 加载数据字节 (9)	0010011_11**iiiiii**	xxxxxxx_xxxxxxxx	(4)
7c. 写锁定位	0110011_00000000 0110001_00000000 0110011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	(1)
7d. 检测写锁定位是否结束	0110011_00000000	xxxxx**o**x_xxxxxxxx	(2)
8a. 进入熔丝位 / 锁定位读操作	0100011_00000100	xxxxxxx_xxxxxxxx	
8b. 读扩展熔丝位字节 (6)	0111010_00000000 0111011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8c. 读熔丝位高字节 (7)	0111110_00000000 0111111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8d. 读熔丝位低字节 (8)	0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
8e. 读锁定位 (9)	0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xx**oooooo**	(5)

Table 130. JTAG 编程指令集
a = 高位地址 , **b** = 低位地址 , **H** = 0 - 高字节 , 1 - 低字节 , **o** = 数据输出 , **i** = 数据输入 , x = 无用途

指令	TDI 序列	TDO 序列	注意
8f. 读熔丝位及锁定位	0111010_00000000 0111110_00000000 0110010_00000000 0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo** xxxxxxx_**oooooooo** xxxxxxx_**oooooooo** xxxxxxx_**oooooooo**	(5) fuse ext. byte fuse high byte fuse low byte lock bits
9a. 进入标识字节读操作	0100011_00001000	xxxxxxx_xxxxxxxx	
9b. 加载地址字节	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
9c. 读标识字节	0110010_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
10a. 进入校验字节读操作	0100011_00001000	xxxxxxx_xxxxxxxx	
10b. 加载地址字节	0000011_**bbbbbbbb**	xxxxxxx_xxxxxxxx	
10c. 读校验字节	0110110_00000000 0110111_00000000	xxxxxxx_xxxxxxxx xxxxxxx_**oooooooo**	
11a. 加载无操作命令	0100011_00000000 0110011_00000000	xxxxxxx_xxxxxxxx xxxxxxx_xxxxxxxx	

Notes:
1. 如果在前一个命令序列 (正常情况) 中已正确设置了 7 个 MSB，那么就不需要这个命令序列。
2. 重复到 o = “1”。
3. 相应的熔丝位 “0” = 编程 “1” = 不编程。
4. 相应的锁定位 “0” = 编程 “1” = 不编程。
5. “0” = 编程 “1” = 不编程。
6. 对应的扩展熔丝位字节的位映射列于 P 269Table 117 。
7. 对应的熔丝位高字节的位映射列于 P 270Table 118 。
8. 对应的熔丝位低字节的位映射列于 P 270Table 119 。
9. 对应的锁定位字节的位映射列于 P 268Table 115 。
10. 忽略超过 PCMSB 及 EEAMSB(Table 123 及 Table 124) 的地址。



Figure 149. 改变 / 读取数据字的状态机序列

虚拟 Flash 页面加载寄存器

虚拟 Flash 页加载寄存器是长度等于一页 Flash 的虚拟扫描链。内部移位寄存器为 8 位，数据自动以字节为单位传入 Flash 页缓冲器。在页中移入所有指令字，由页内第一条指令的 LSB 开始，到最后一条指令的 MSB 结束。这为在执行页写操作前加载整个 Flash 页缓冲器提供了一条有效的措施。

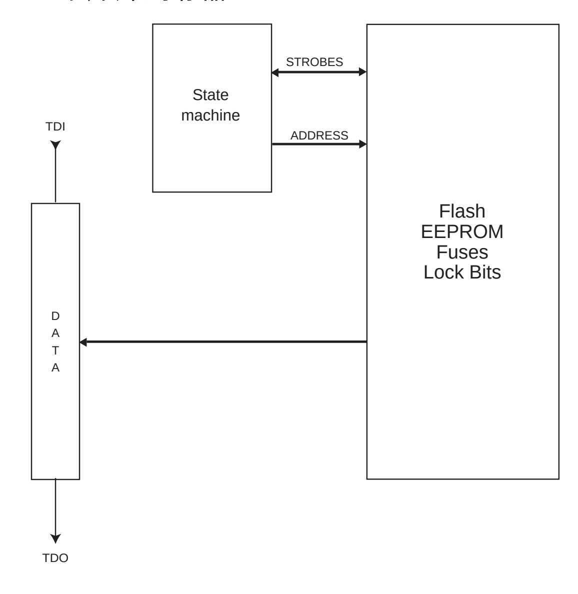
Figure 150. 虚拟 Flash 页加载寄存器

State machine
STROBES
ADDRESS
TDI
Flash
EEPROM
Fuses
Lock Bits
DATA
TDO

虚拟 Flash 页面读取寄存器

虚拟 Flash 页读取寄存器是长度等于一页 Flash 加 8 的虚拟扫描链。内部移位寄存器为 8 位 , 数据自动以字节为单位从 Flash 数据页传出。第一个 8 周期用来传送内部移位寄存器的第一个字节 , 且该字节应被忽略。在这一初始化后 , 数据移出由页内第一条指令的 LSB 开始 , 到最后一条指令的 MSB 结束。这为读取整个 Flash 页校验编程提供了一条有效的措施。

Figure 151. 虚拟 Flash 页读取寄存器



编程算法

所有类似 “1a”， “1b” 这样的参考请参见 Table 130。

进入编程模式

1. 进入 JTAG 指令 AVR_RESET，并把 1 移入 Reset 寄存器。
2. 进入 PROG_ENABLE 指令，并把 1010_0011_0111_0000 送入编程使能寄存器。

退出编程模式

1. 进入 JTAG 指令 PROG_COMMANDS。
2. 通过无操作指令 11a 来禁止所有编程指令。

3. 进入 PROG_ENABLE 指令 , 并将 0000_0000_0000_0000 送入编程使能寄存器。
4. 进入 JTAG 指令 AVR_RESET，并把 0 送入 Reset 寄存器。

执行芯片擦除操作

1. 进入 JTAG 指令 PROG_COMMANDS。
2. 使用编程指令 1a 进行芯片擦除。
3. 使用 1b 指令检查芯片擦除是否完成，或者等待 t_{WLRH_CE}(见 P 280Table Note:)。

对 Flash 进行编程

在对 Flash 编程前必须执行芯片擦除，见 P 296“ 执行芯片擦除操作 ” 。

1. 进入 JTAG 指令 PROG_COMMANDS。
2. 使用编程指令 2a 启动 Flash 写操作。
3. 使用编程指令 2b 加载地址高字节。
4. 使用编程指令 2c 加载地址低字节。
5. 使用编程指令 2d、2e 及 2f 加载数据。
6. 对这一页的所有程序字重复操作 4 和操作 5。
7. 使用编程指令 2g 进行页写操作。
8. 使用编程指令 2h 检查 Flash 编程是否结束 , 或等待 t_{WLRH}(见 P 280Table Note:)。
9. 重复操作 3 到 7，直到所有的数据都被编程。

可通过使用 PROG_PAGELOAD 指令来得到更高的数据传输率：

1. 进入 JTAG 指令 PROG_COMMANDS。
2. 通过使用编程指令 2a 启动 Flash 写操作。
3. 使用编程指令 2b 及 2c 加载页地址。PCWORD(见 P 273Table 123) 用于页内寻址，它必须为 0。
4. 进入 JTAG 指令 PROG_PAGELOAD。
5. 一个字节一个字节地将页内的程序字加载到整个页，由第一条指令的 LSB 开始，结束于最后一条指令的 MSB。
6. 进入 JTAG 指令 PROG_COMMANDS。
7. 使用编程指令 2g 执行页写操作。
8. 使用编程指令2h检查Flash写操作是否完成 , 或等待t_{WLRH}(见 P 280Table Note:)。
9. 重复步骤 3 到 8 ，直到所有的数据都被编程。

读取 Flash

1. 进入 JTAG 指令 PROG_COMMANDS。
2. 使用编程指令 3a 启动 Flash 读操作。
3. 使用编程指令 3b 及 3c 加载地址。
4. 使用编程指令 3d 读数据。
5. 重复操作 3 和 4，直到所有数据都被读出。

通过使用 PROG_PAGEREAD 指令可以更高效地传输数据：

1. 进入 JTAG 指令 PROG_COMMANDS。
2. 使用编程指令 3a 启动 Flash 读操作。
3. 使用编程指令 3b 及 3c 来加载页地址。PCWORD(见 P 273Table 123) 用于页内寻址，必须为 0。
4. 进入 JTAG 指令 PROG_PAGEREAD。
5. 通过将一页之中的所有程序字移出来读取一整页，由第一条指令的 LSB 开始，由终止于最后一条指令的 MSB。第一个移出的字节要忽略。
6. 使用 JTAG 指令 PROG_COMMANDS。
7. 重复步骤 3 到 6，直到所有数据都被读出。

对 EEPROM 进行编程

在对 EEPROM 编程前，必须执行芯片擦除，见 P 296“ 执行芯片擦除操作 ” 。

1. 进入 JTAG 指令 PROG_COMMANDS。
2. 使用编程指令 4a 启动 EEPROM 写操作。
3. 使用编程指令 4b 来加载地址高字节。
4. 使用编程指令 4c 来加载地址低字节。
5. 使用 4d 及 4e 加载数据。
6. 对页内所有数据字节执行操作 4 和 5。
7. 使用编程指令 4f 写数据。
8. 使用编程指令4g检查EEPROM写操作是否已经完成，或等待t_{WLRH}(见 P 280Table Note:)。
9. 重复步骤 3 到 8，直到所有的数据都被编程。

编程 EEPROM 时不能使用 PROG_PAGELOAD 指令。

读取 EEPROM

1. 进入 JTAG 指令 PROG_COMMANDS。
2. 使用编程指令 5a 启动 EEPROM 读操作。
3. 使用编程指令 5b 及 5c 加载地址。
4. 使用编程指令 5d 读数据。
5. 重复操作 3 和 4，直到所有数据都被读出。

注意，读 EEPROM 时不能使用 PROG_PAGEREAD 指令。



对熔丝位进行编程

1. 进入 JTAG 指令 PROG_COMMANDS。
2. 使用编程指令 6a 启动熔丝位写操作。
3. 使用编程指令 6b 加载数据高字节。位值为 0 表示相应熔丝位需要编程，否则不编程。
4. 使用编程指令 6c 写熔丝位高字节。
5. 使用编程指令 6d 检查熔丝位写操作是否完成，或等待 t_{WLRH}(见 P 280Table Note:)。
6. 使用编程指令 6e 加载数据字节，写 0 表示熔丝位编程，写 1 表示未编程。
7. 使用编程指令 6f 写熔丝位高字节。
8. 使用编程指令 6g 检查熔丝位写操作是否完成，或等待 t_{WLRH}(见 P 280Table Note:)。
9. 使用编程指令 6h 加载数据字节，写 0 表示熔丝位编程，写 1 表示未编程。
10. 使用编程指令 6i 写熔丝位低字节。
11. 使用编程指令 6j 检查熔丝位写操作是否完成，或等待 t_{WLRH}(见 P 280Table Note:)。

对锁定位进行编程

1. 进入 JTAG 指令 PROG_COMMANDS。
2. 使用编程指令 7a 进入锁定位写操作。
3. 使用编程指令 7b 进行数据加载。位值为 0 表示相应熔丝位需要编程，否则不编程。
4. 使用编程指令 7c 写锁定位。
5. 使用编程指令 7d 检验锁定位写操作是否完成，或等待 t_{WLRH}(见 P 280Table Note:)。

读取熔丝位和锁定位

1. 进入 JTAG 指令 PROG_COMMANDS。
2. 使用编程指令 8a 进入熔丝位 / 锁定位读操作。
3. 使用编程指令 8f 来读取所有熔丝位及锁定位。
   使用编程指令 8b 来读取扩展熔丝位字节。
   使用编程指令 8c 来读取熔丝位高字节。
   使用编程指令 8d 来读取熔丝位低字节。
   使用编程指令 8e 读取锁定位。

读取标识字节

1. 进入 JTAG 指令 PROG_COMMANDS。
2. 使用指令 9a 启动标识字节读操作。
3. 使用编程指令 9b 加载地址 $00。
4. 使用编程指令 9c 读第一个标识字节。
5. 在地址 $01 及 $02 处重复操作 3 和 4，分别读第二及第三个标识字节。

读取标定字节

1. 进入 JTAG 指令 PROG_COMMANDS。
2. 使用 10a 指令启动检验字节读操作。
3. 使用编程指令 10b 加载地址 $00。
4. 使用编程指令 10c 读取校验字节。

电气特性

Note: 典型数据基于仿真以及其他使用相同工艺生产的 AVR 微处理器的特性数据。对器件进行特性化之后将给出最大值和最小值。

绝对极限值 *

工作温度 ....-55°C 到 +125°C

存储温度 ....-65°C 到 +150°C

引脚上的电压，除了 $\overline{RESET}$
相对于地 .... -1.0V 到 V_{CC}+0.5V

$\overline{RESET}$ 引脚上的电压，相对于地 .... -1.0V 到 +13.0V

最大工作电压 6.0V

每个 I/O 引脚上的 DC 电流 .... 40.0 mA

DC 电流：V_{CC} 和 GND 引脚之间 .... 200.0 mA

*NOTICE: 超出列于此处的绝对极限值可能造成器件的永久损坏。长期工作于绝对极限值之下可能影响器件的可靠性。

直流特性

T_A = -40°C- 85°C, V_{CC} = 2.7V - 5.5V (除非另外说明)

符号	参数	条件	最小值	典型值	最大值	单位
V_{IL}	输入低电压	除了 XTAL1 和 $\overline{RESET}$ 引脚	-0.5		0.3 V_{CC}(1)	V
V_{IL1}	输入低电压	XTAL1 引脚，外部时钟	-0.5		0.1 V_{CC}(1)	V
V_{IL2}	输入低电压	$\overline{RESET}$ 引脚	-0.5		0.2 V_{CC}(1)	V
V_{IH}	输入高电压	除了 XTAL1 和 $\overline{RESET}$ 引脚	0.6 V_{CC}(2)		V_{CC} + 0.5	V
V_{IH1}	输入高电压	XTAL1 引脚，外部时钟	0.7 V_{CC}(2)		V_{CC} + 0.5	V
V_{IH2}	输入高电压	$\overline{RESET}$ 引脚	0.85 V_{CC}(2)		V_{CC} + 0.5	V
V_{OL}	输出低电压 (3) (端口 A,B,C,D, E, F, G)	I_{OL} = 20 mA, V_{CC} = 5V I_{OL} = 10 mA, V_{CC} = 3V			0.7 0.5	V V
V_{OH}	输出高电压 (端口 A,B,C,D)	I_{OH} = -20 mA, V_{CC} = 5V I_{OH} = -10 mA, V_{CC} = 3V	4.0 2.2			V V
I_{IL}	输入泄露电流 I/O 引脚	Vcc = 5.5V, 引脚为低电平 (绝对值)			8.0	µA
I_{IH}	输入泄露电流 I/O 引脚	Vcc = 5.5V, 引脚为高电平 (绝对值)			8.0	µA
R_{RST}	Reset 引脚上拉电阻		30		100	kΩ
R_{PEN}	PEN 引脚上拉电阻		25		100	kΩ
R_{PU}	I/O 引脚上拉电阻		33		122	kΩ



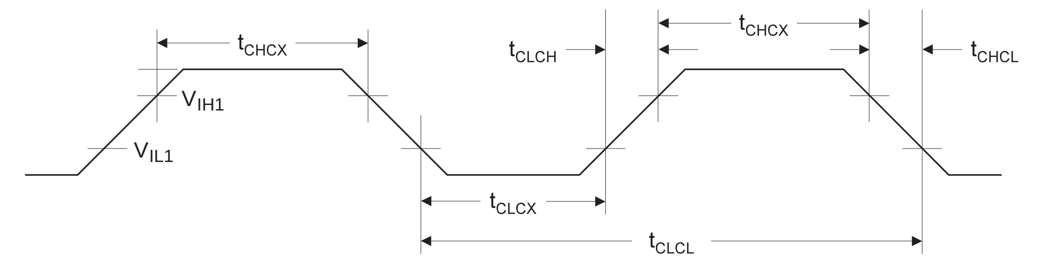
T_A = -40°C- 85°C, V_{CC} = 2.7V - 5.5V (除非另外说明)

符号	参数	条件	最小值	典型值	最大值	单位
I_{CC}	工作电流	4 MHz, V_{CC} = 3V (ATmega128L)			5	mA
		8 MHz, V_{CC} = 5V (ATmega128)			20	mA
		空闲，4 MHz, V_{CC} = 3V (ATmega128L)			2	mA
		空闲， 8 MHz, V_{CC} = 5V (ATmega128)			12	mA
	掉电模式 [5]	WDT 使能，V_{CC} = 3V		< 25	40	µA
		WDT 禁止，V_{CC} = 3V		< 10	25	µA
V_{ACIO}	模拟比较器 输入偏置电压	V_{CC} = 5V V_{in} = V_{CC}/2			40	mV
I_{ACLK}	模拟比较器 输入泄漏电流	V_{CC} = 5V V_{in} = V_{CC}/2	-50		50	nA
t_{ACID}	模拟比较器 初始化延迟	V_{CC} = 2.7V V_{CC} = 5.0V		750 500		ns

Notes:
1. “ 最大值 ” 表示保证引脚读取数值为低时的最高值。
2. “ 最小值 ” 表示保证引脚读取数值为高时的最低值。
3. 虽然在稳定状态条件(非瞬态)下每个I/O端口都可以吸收比测试条件下更多的电流(20 mA ,V_{CC} = 5V 以及10 mA，V_{CC} = 3V) ，但是需要遵循以下要求：
   TQFP 封装：
   1] 所有端口的 IOL 总和不能超过 400 mA。
   2] 端口 A0 - A7, G2, C3 - C7 的 IOL 总和不能超过 300 mA。
   3] 端口 C0 - C2, G0 - G1, D0 - D7, XTAL2 的 IOL 总和不能超过 150 mA。
   4] 端口 B0 - B7, G3 - G4, E0 - E7 的 IOL 总和不能超过 150 mA。
   5] 端口 F0 - F7 的 IOL 总和不能超过 200 mA。
   如果 IOL 超出了测试条件， VOL 可能超过指标。不保证引脚可以吸收比列于此处的测试条件更大的电流。
4. 虽然在稳定状态条件(非瞬态)下每个I/O端口都可以输出比测试条件下更多的电流(20 mA ,V_{CC} = 5V 以及10 mA，V_{CC} = 3V) ，但是需要遵循以下要求：
   TQFP 封装：
   1] 所有端口的 IOH 总和不能超过 400 mA。
   2] 端口 A0 - A7, G2, C3 - C7 的 IOL 总和不能超过 300 mA。
   3] 端口 C0 - C2, G0 - G1, D0 - D7, XTAL2 的 IOL 总和不能超过 150 mA。
   4] 端口 B0 - B7, G3 - G4, E0 - E7 的 IOL 总和不能超过 150 mA。
   5] 端口 F0 - F7 的 IOL 总和不能超过 200 mA。
   如果 IOH 超出了测试条件， VOH 可能超过指标。不保证引脚可以输出比列于此处的测试条件更大的电流。

外部时钟波形

Figure 152. 外部时钟驱动波形



外部时钟

Table 131. 外部时钟驱动

符号	参数	V_{CC} = 2.7V - 5.5V		V_{CC} = 4.5V - 5.5V		单位
		最小值	最大值	最小值	最大值	
$1/t_{CLCL}$	振荡器频率	0	8	0	16	MHz
t_{CLCL}	时钟周期	125		62.5		ns
t_{CHCX}	高电平时间	50		25		ns
t_{CLCX}	低电平时间	50		25		ns
t_{CLCH}	上升时间		1.6		0.5	μs
t_{CHCL}	下降时间		1.6		0.5	μs
Δt_{CLCL}	参数		2		2	%

Table 132. 外部 RC 振荡器，典型频率

R [kΩ][1]	C [pF]	f[2]
100	47	87 kHz
33	22	650 kHz
10	22	2.0 MHz

Notes: 1. R 的取值范围为 3 kΩ - 100 kΩ ,C 至少应该为 20 pF。此处给出的 C 的数值包括引脚电容。引脚电容随封装形式而变化。
2. 频率随封装与板层的不同而变化。

两线串行接口特性

Table 133 描述了连接到两线串行总线上的器件的要求。 ATmega128 的两线接口满足或超出此处列出的要求。

时序符号请参考 Figure 153。

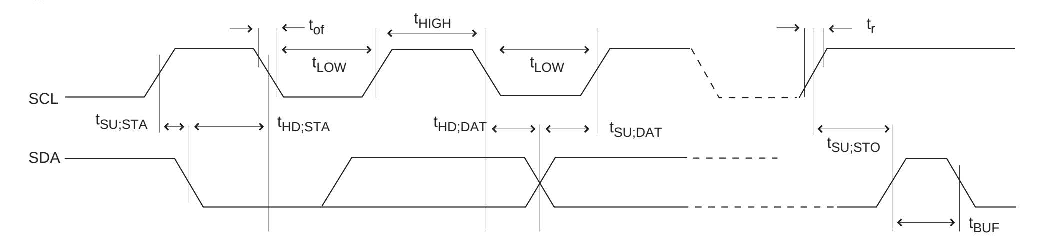
Table 133. 两线串行总线要求

符号	参数	条件	最小值	最大值	单位
V_{IL}	输入低电压		-0.5	0.3 V_{CC}	V
V_{IH}	输入高电压		0.7 V_{CC}	V_{CC} + 0.5	V
V_{hys}(1)	施密特触发器输入的迟滞电压		0.05 V_{CC}(2)	–	V
V_{OL}(1)	输出入低电压	3 mA 漏电流	0	0.4	V
t_r(1)	SDA 和 SCL 的上升时间		20 + 0.1C_b(3)(2)	300	ns
t_{of}(1)	由 V_{IHmin} 到 V_{ILmax} 的输出下降时间	10 pF < C_b < 400 pF(3)	20 + 0.1C_b(3)(2)	250	ns
t_{SP}(1)	输入滤波器抑制的尖峰时间		0	50(2)	ns
I_i	每个 I/O 引脚的输入电流	0.1 V_{CC} < V_i < 0.9 V_{CC}	-10	10	µA
C_i(1)	每个 I/O 引脚的电容		–	10	pF
f_{SCL}	SCL 时钟频率	f_{CK}(4) > max(16f_{SCL}, 250kHz)(5)	0	400	kHz
Rp	上拉电阻值	$f_{SCL} \le$ 100 kHz	$\frac{V_{CC} - 0{,}4V}{3mA}$	$\frac{1000ns}{C_b}$	Ω
		f_{SCL} > 100 kHz	$\frac{V_{CC} - 0{,}4V}{3mA}$	$\frac{300ns}{C_b}$	Ω
$t_{HD;STA}$	START 条件的保持时间 (重复)	$f_{SCL} \le$ 100 kHz	4.0	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
t_{LOW}	SCL 时钟的低电平时间	$f_{SCL} \le$ 100 kHz(6)	4.7	–	µs
		f_{SCL} > 100 kHz(7)	1.3	–	µs
t_{HIGH}	SCL 时钟的高电平时间	$f_{SCL} \le$ 100 kHz	4.0	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
$t_{SU;STA}$	重复 STARTS 条件的建立时间	$f_{SCL} \le$ 100 kHz	4.7	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
$t_{HD;DAT}$	数据保持时间	$f_{SCL} \le$ 100 kHz	0	3.45	µs
		f_{SCL} > 100 kHz	0	0.9	µs
$t_{SU;DAT}$	数据建立时间	$f_{SCL} \le$ 100 kHz	250	–	ns
		f_{SCL} > 100 kHz	100	–	ns
$t_{SU;STO}$	STOP 条件的建立时间	$f_{SCL} \le$ 100 kHz	4.0	–	µs
		f_{SCL} > 100 kHz	0.6	–	µs
t_{BUF}	STOP 和 START 之间的总线空闲时间	$f_{SCL} \le$ 100 kHz	4.7	–	µs

Notes:
1. 对于 ATmega128，此参数是特性参数，没有经过 100% 的测试。
2. 只有当 f_{SCL} > 100 kHz 时才需要。
3. C_b = 总线的一条线的电容。
4. f_{CK} = CPU 时钟频率。
5. 此要求适用于 ATmega128 所有的两线串行接口的操作。其他连接到两线串行总线的器件只需要满足一般的 f_{SCL} 要求即可。

6. ATmega128两线串行接口实际产生的低电平时间为 $(1/f_{SCL} - 2/f_{CK})$。因此为了严格满足 f_{SCL} = 100 kHz 时低电平时间的要求 f_{CK} 必须大于 6 MHz。
7. ATmega128 两线串行接口实际产生的低电平时间为 $(1/f_{SCL} - 2/f_{CK})$。因此在 f_{CK} = 8 MHz，且 f_{SCL} > 308 kHz 时低电平时间无法严格满足要求。然而，ATmega128 可以与其他 ATmega128 以全速 (400 kHz) 进行通讯。若其他器件具有合适的 t_{LOW} 接受裕量也可以做到这一点。

Figure 153. 两线串行总线时序



SPI 时序特性

具体信息请参见 Figure 154 和 Figure 155。

Table 134. SPI 时序参数

	说明	模式	最小值	典型值	最大值	
1	SCK 周期	主机		参见 Table 72		
2	SCK 高 / 低电平	主机		50% 占空比		
3	上升 / 下降时间	主机		TBD		
4	建立时间	主机		10		
5	保持时间	主机		10		
6	输出到 SCK	主机		$0.5 \cdot t_{sck}$		ns
7	SCK 到输出	主机		10		
8	SCK 到输出高电平	主机		10		
9	SS 低到输出	从机		15		
10	SCK 周期	从机	$4 \cdot t_{ck}$			
11	SCK 高 / 低电平	从机	$2 \cdot t_{ck}$			
12	上升 / 下降时间	从机		TBD		μs
13	建立时间	从机	10			
14	保持时间	从机	10			
15	SCK 到输出	从机		15		
16	SCK 到 $\overline{SS}$ 高	从机	20			ns
17	$\overline{SS}$ 高到三态	从机		10		
18	SS 低到 SCK	从机	20			

Note: 1. SPI 编程模式下 SCK 最小周期为：
- 2 t_{CLCL} for f_{CK} < 12 MHz
- 3 t_{CLCL} for f_{CK} >12 MHz

Figure 154. SPI 接口时序要求 (主机模式)

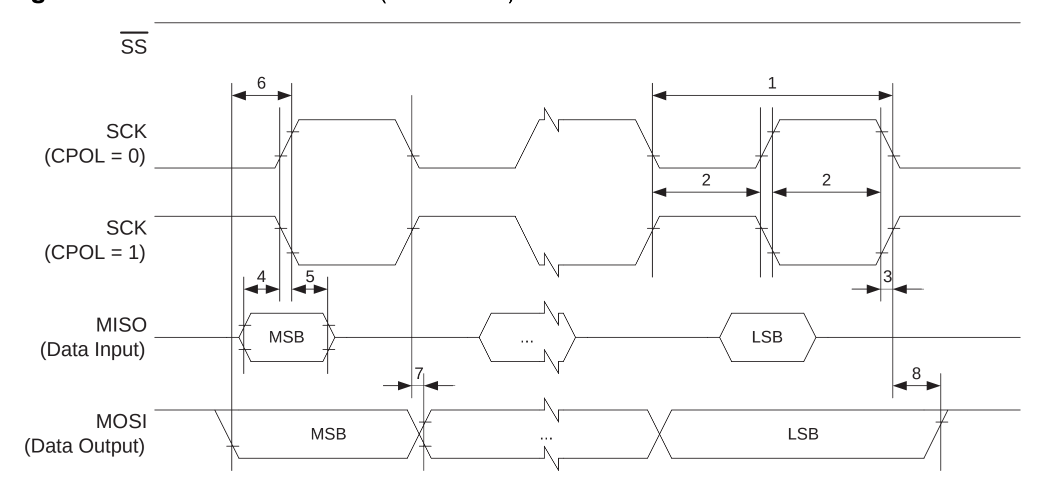
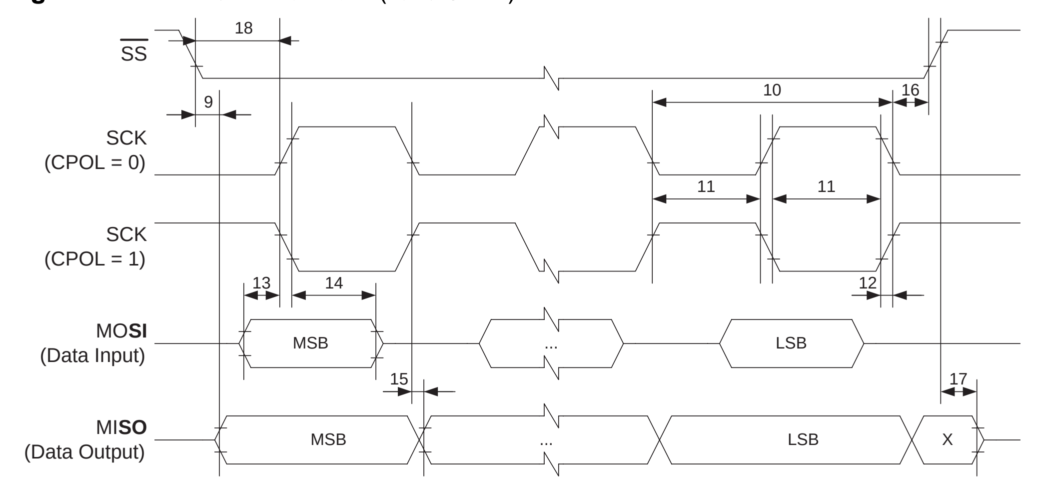


Figure 155. SPI 接口时序要求 (从机模式)



交流特性

Table 135. ADC 特性参数，单端通道

符号	参数	条件	最小值 [1]	典型值 [1]	最大值 [1]	单位
	分辨率	单极性转换			10	Bits
		差分方式 增益 = 1x 或 20x		1.5		LSB
		差分方式 增益 = 200x		3.25		LSB
	绝对精度	单极性转换 V_{REF} = 4V ADC 时钟 = 200 kHz ADHSM = 0		1.5		LSB
		单极性转换 V_{REF} = 4V ADC 时钟 = 1 MHz ADHSM = 1		3.75		LSB
	积分非线性	V_{REF} = 4V		0.75		LSB
	差分非线性	V_{REF} = 4V		0.5		LSB
	零误差（偏置）	V_{REF} = 4V		1		LSB
	转换时间	自由运行方式 ADHSM = 0		1		LSB
		自由运行方式 ADHSM = 1	50		1000	kHz
	时钟频率	ADHSM = 0	13		260	μs
AVCC		ADHSM = 1	V_{CC} - 0.3 [2]		V_{CC} + 0.3 [3]	V
V_{REF}	模拟电压		2.0		AVCC	V
V_{IN}	参考电压	单极性转换	GND		V_{REF}	V
		差分方式			38.5	kHz
V_{INT}	输入电压	单端通道	2.3	2.56	2.7	V
R_{REF}		差分通道		32		kΩ
R_{AIN}	输入带宽	单端通道	55	100		MΩ

Notes: 1. 数值仅作为参考。实际数据待定。
2. AVCC 的最小值为 2.7 V。
3. AVCC 的最大值为 5.5 V。





Table 136. ADC 特性参数，差分通道

符号	参数	条件	最小值 (1)	典型值 (1)	最大值 (1)	单位
	分辨率	增益 = 1x			10	Bits
		增益 = 10x			10	Bits
		增益 = 200x			10	Bits
	绝对精度	增益 = 1x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		17		LSB
		增益 = 10x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		17		LSB
		增益 = 200x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		7		LSB
	积分非线性 (INL) (精度为补偿与增益误差标定后的值)	增益 = 1x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		1.5		LSB
		增益 = 10x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		2		LSB
		增益 = 200x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		5		LSB
	增益误差	增益 = 1x		1.5		%
		增益 = 10x		1.5		%
		增益 = 200x		0.5		%
	补偿误差	增益 = 1x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		2		LSB
		增益 = 10x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		3		LSB
		增益 = 200x V_{REF} = 4V, V_{CC} = 5V ADC 时钟 = 50 - 200 kHz		4		LSB
	时钟频率		50		200	kHz
	转换时间		65		260	µs
AVCC	模拟电压		V_{CC} - 0.3[2]		V_{CC} + 0.3[3]	V
V_{REF}	参考电压		2.0		AVCC - 0.5	V
V_{IN}	输入电压		GND		V_{CC}	V
V_{DIFF}	输入差分电压		-V_{REF}/Gain		V_{REF}/Gain	V
	ADC 转换输出		-511		511	LSB
	输入带宽			4		kHz

Table 136. ADC 特性参数，差分通道 (Continued)

符号	参数	条件	最小值 [1]	典型值 [1]	最大值 [1]	单位
V_{INT}	内部电压基准		2.3	2.56	2.7	V
R_{REF}	参考输入端电阻			32		kΩ
R_{AIN}	模拟输入电阻		55	100		MΩ

Notes: 1. 数值仅作为参考。实际数据待定。
2. AVCC 的最小值为 2.7 V。
3. AVCC 的最大值为 5.5 V。

外部数据存储器时序

Table 137. 外部数据存储器特性参数， 4.5 - 5.5V，无等待状态

			8 MHz 振荡器		其他振荡器		
	符号	参数	最小值	最大值	最小值	最大值	单位
0	$1/t_{CLCL}$	振荡器频率			0.0	16	MHz
1	t_{LHLL}	ALE 脉冲宽度	115		$1.0t_{CLCL}$-10		ns
2	t_{AVLL}	地址有效到 ALE 为低	57.5		$0.5t_{CLCL}$-5(1)		ns
3a	t_{LLAX_ST}	ALE 为低之后的地址保持时间 写操作	5		5		ns
3b	t_{LLAX_LD}	为低之后的地址保持时间 读操作	5		5		ns
4	t_{AVLLC}	地址有效 C 到 ALE 低	57.5		$0.5t_{CLCL}$-5(1)		ns
5	t_{AVRL}	地址有效到 RD 为低	115		$1.0t_{CLCL}$-10		ns
6	t_{AVWL}	地址有效到 WR 为低	115		$1.0t_{CLCL}$-10		ns
7	t_{LLWL}	ALE 低到 WR 低	47.5	67.5	$0.5t_{CLCL}$-15(2)	$0.5t_{CLCL}$+5(2)	ns
8	t_{LLRL}	ALE 低到 RD 低	47.5	67.5	$0.5t_{CLCL}$-15(2)	$0.5t_{CLCL}$+5(2)	ns
9	t_{DVRH}	数据建立到 RD 为高	40		40		ns
10	t_{RLDV}	读信号低到数据有效		75		$1.0t_{CLCL}$-50	ns
11	t_{RHDX}	RD 为高之后的数据保持时间	0		0		ns
12	t_{RLRH}	RD 脉冲宽度	115		$1.0t_{CLCL}$-10		ns
13	t_{DVWL}	数据建立到 WR 为低	42.5		$0.5t_{CLCL}$-20(1)		ns
14	t_{WHDX}	WR 为高之后的数据保持时间	115		$1.0t_{CLCL}$-10		ns
15	t_{DVWH}	数据有效到 WR 为高	125		$1.0t_{CLCL}$		ns
16	t_{WLWH}	WR 脉冲宽度	115		$1.0t_{CLCL}$-10		ns

Notes: 1. 假定时钟占空比为 50%。一半周期为外部时钟 XTAL1 的实际高电平时间。
2. 假定时钟占空比为 50%。一半周期为外部时钟 XTAL1 的实际低电平时间。

Table 138. 外部数据存储器特性参数， 4.5 - 5.5V，一个等待状态

			8 MHz 振荡器		其他振荡器		
	符号	参数	最小值	最大值	最小值	最大值	单位
0	$1/t_{CLCL}$	振荡器频率			0.0	16	MHz
10	t_{RLDV}	读信号低到数据有效		200		$2.0t_{CLCL}$-50	ns
12	t_{RLRH}	RD 脉冲宽度	240		$2.0t_{CLCL}$-10		ns
15	t_{DVWH}	数据有效到 WR 为高	240		$2.0t_{CLCL}$		ns
16	t_{WLWH}	WR 脉冲宽度	240		$2.0t_{CLCL}$-10		ns

Table 139. 外部数据存储器特性参数， 4.5 - 5.5V， SRWn1 = 1， SRWn0 = 0

	符号	参数	4 MHz 振荡器		其他振荡器		单位
			最小值	最大值	最小值	最大值	
0	$1/t_{CLCL}$	振荡器频率			0.0	16	MHz
10	t_{RLDV}	读信号低到数据有效		325		$3.0t_{CLCL}$-50	ns
12	t_{RLRH}	RD 脉冲宽度	365		$3.0t_{CLCL}$-10		ns
15	t_{DVWH}	数据有效到 WR 为高	375		$3.0t_{CLCL}$		ns
16	t_{WLWH}	WR 脉冲宽度	365		$3.0t_{CLCL}$-10		ns

Table 140. 外部数据存储器特性参数， 4.5 - 5.5V， SRWn1 = 1， SRWn0 = 1

	符号	参数	4 MHz 振荡器		其他振荡器		单位
			最小值	最大值	最小值	最大值	
0	$1/t_{CLCL}$	振荡器频率			0.0	16	MHz
10	t_{RLDV}	读信号低到数据有效		325		$3.0t_{CLCL}$-50	ns
12	t_{RLRH}	RD 脉冲宽度	365		$3.0t_{CLCL}$-10		ns
14	t_{WHDX}	WR 为高之后数据保持时间	240		$2.0t_{CLCL}$-10		ns
15	t_{DVWH}	数据有效到 WR 为高	375		$3.0t_{CLCL}$		ns
16	t_{WLWH}	WR 脉冲宽度	365		$3.0t_{CLCL}$-10		ns

Table 141. 外部数据存储器特性参数， 2.7 - 5.5V，无等待状态

	符号	参数	4 MHz 振荡器		其他振荡器		单位
			最小值	最大值	最小值	最大值	
0	$1/t_{CLCL}$	振荡器频率			0.0	8	MHz
1	t_{LHLL}	ALE 脉冲宽度	235		t_{CLCL}-15		ns
2	t_{AVLL}	ALE 为低地址有效 A	115		$0.5t_{CLCL}$-10(1)		ns
3a	t_{LLAX_ST}	ALE 为低后地址保持，写访问	5		5		ns
3b	t_{LLAX_LD}	ALE 为低后地址保持，读访问	5		5		ns
4	t_{AVLLC}	ALE 为低地址有效 C	115		$0.5t_{CLCL}$-10(1)		ns
5	t_{AVRL}	RD 为低地址有效	235		$1.0t_{CLCL}$-15		ns
6	t_{AVWL}	WR 为低地址有效	235		$1.0t_{CLCL}$-15		ns
7	t_{LLWL}	ALE 为低 WR 为低	115	130	$0.5t_{CLCL}$-10(2)	$0.5t_{CLCL}$+5(2)	ns
8	t_{LLRL}	ALE 为低 RD 为低	115	130	$0.5t_{CLCL}$-10(2)	$0.5t_{CLCL}$+5(2)	ns
9	t_{DVRH}	RD 为高数据建立	45		45		ns
10	t_{RLDV}	读低数据有效		190		$1.0t_{CLCL}$-60	ns
11	t_{RHDX}	RD 为高后数据保持	0		0		ns



Table 141. 外部数据存储器特性参数， 2.7 - 5.5V，无等待状态

	符号	参数	4 MHz 振荡器		其他振荡器		单位
			最小值	最大值	最小值	最大值	
12	t_{RLRH}	RD 脉冲宽带	235		$1.0t_{CLCL}$-15		ns
13	t_{DVWL}	WR 为低数据建立	105		$0.5t_{CLCL}$-20(1)		ns
14	t_{WHDX}	WR 为高后数据保持	235		$1.0t_{CLCL}$-15		ns
15	t_{DVWH}	WR 为高数据有效	250		$1.0t_{CLCL}$		ns
16	t_{WLWH}	WR 脉冲宽带	235		$1.0t_{CLCL}$-15		ns

Notes: 1. 假设占空比为 50%，半周期实际为外部时钟 XTAL1 的高时间。
2. 假设占空比为 50%，半周期实际为外部时钟 XTAL1 的低时间。

Table 142. 外部数据存储器特性， 2.7 - 5.5 V， SRWn1 = 0， SRWn0 = 1

	符号	参数	4 MHz 振荡器		可变频振荡器		单位
			最小值	最大值	最小值	最大值	
0	$1/t_{CLCL}$	振荡器频率			0.0	8	MHz
10	t_{RLDV}	读低时数据有效		440		$2.0t_{CLCL}$-60	ns
12	t_{RLRH}	RD 脉冲宽度	485		$2.0t_{CLCL}$-15		ns
15	t_{DVWH}	WR 高数据有效	500		$2.0t_{CLCL}$		ns
16	t_{WLWH}	WR 脉冲宽度	485		$2.0t_{CLCL}$-15		ns

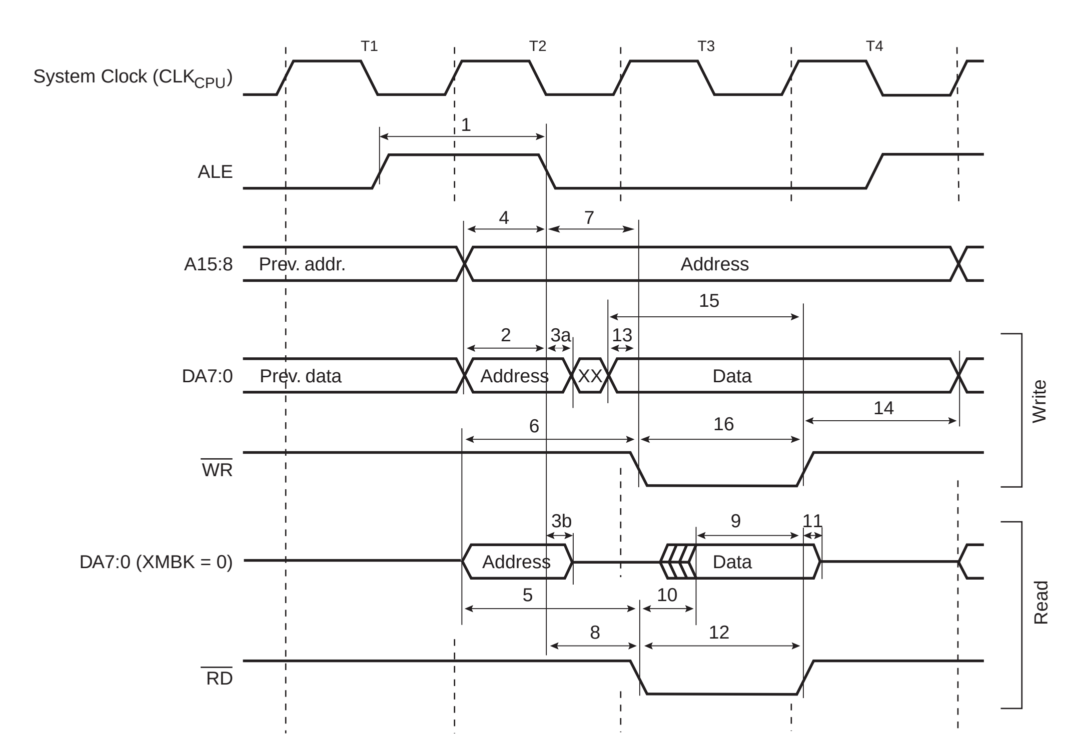
Table 143. 外部数据存储器特性， 2.7 - 5.5 V， SRWn1 = 1， SRWn0 = 0

	符号	参数	4 MHz 振荡器		可变频振荡器		Unit
			最小值	最大值	最小值	最大值	
0	$1/t_{CLCL}$	振荡器频率			0.0	8	MHz
10	t_{RLDV}	读低时数据有效		690		$3.0t_{CLCL}$-60	ns
12	t_{RLRH}	RD 脉冲宽度	735		$3.0t_{CLCL}$-15		ns
15	t_{DVWH}	WR 高数据有效	750		$3.0t_{CLCL}$		ns
16	t_{WLWH}	WR 脉冲宽度	735		$3.0t_{CLCL}$-15		ns

Table 144. 外部数据存储器特性，2.7 - 5.5 V， SRWn1 = 1， SRWn0 = 1

	符号	参数	4 MHz 振荡器		可变频振荡器		单位
			最小值	最大值	最小值	最大值	
0	$1/t_{CLCL}$	振荡器频率			0.0	8	MHz
10	t_{RLDV}	读低时数据有效		690		$3.0t_{CLCL}$-60	ns
12	t_{RLRH}	RD 脉冲宽度	735		$3.0t_{CLCL}$-15		ns
14	t_{WHDX}	WR 为高后数据保持	485		$2.0t_{CLCL}$-15		ns
15	t_{DVWH}	WR 高数据有效	750		$3.0t_{CLCL}$		ns
16	t_{WLWH}	WR 脉冲宽度	735		$3.0t_{CLCL}$-15		ns

Figure 156. 外部存储器时序 (SRWn1 = 0, SRWn0 = 0



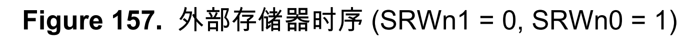


Figure 157. 外部存储器时序 (SRWn1 = 0, SRWn0 = 1)

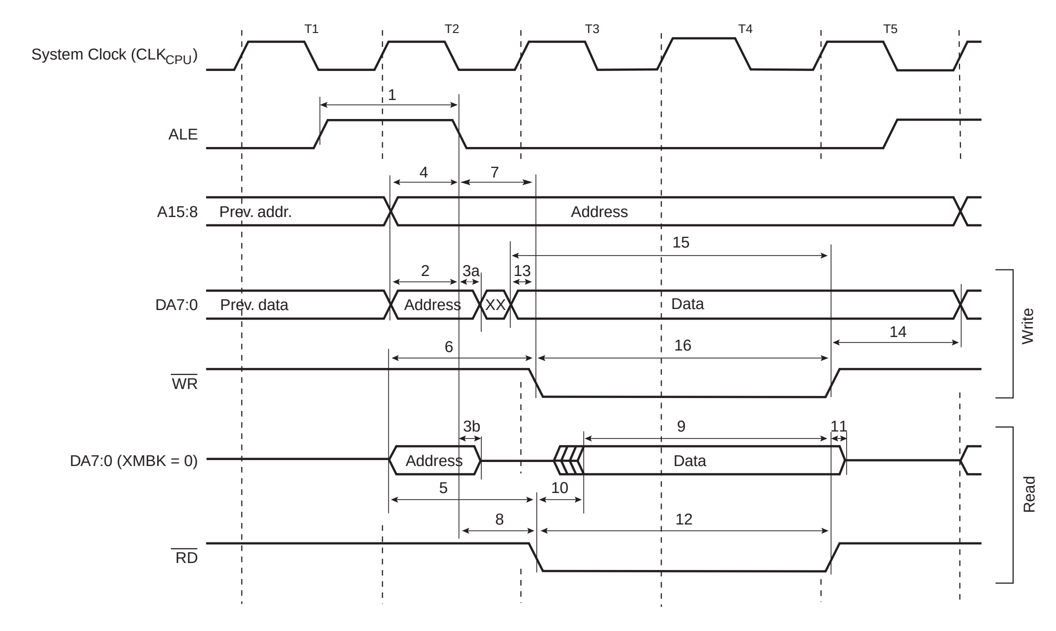




Figure 158. 外部存储器时序 (SRWn1 = 1, SRWn0 = 0)

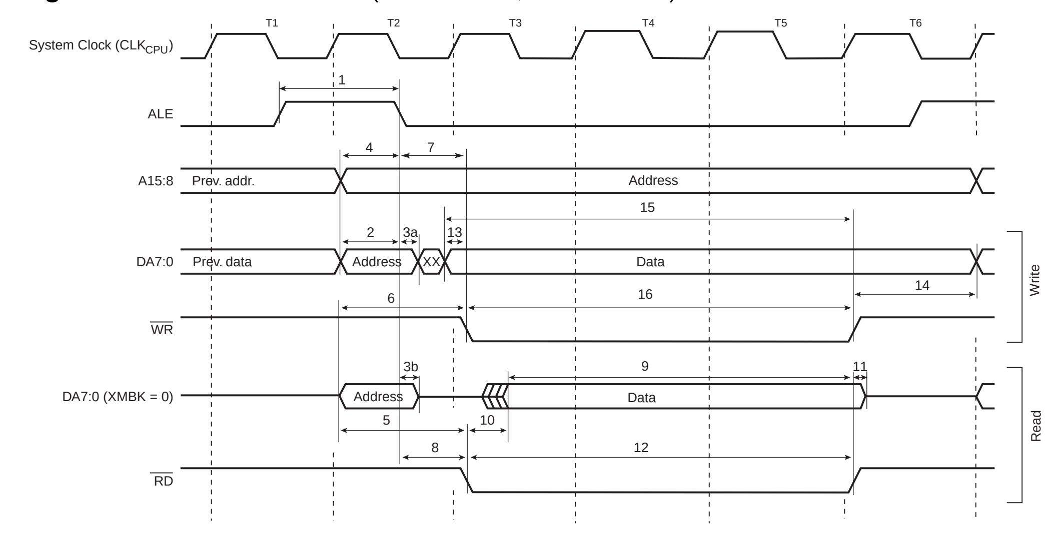
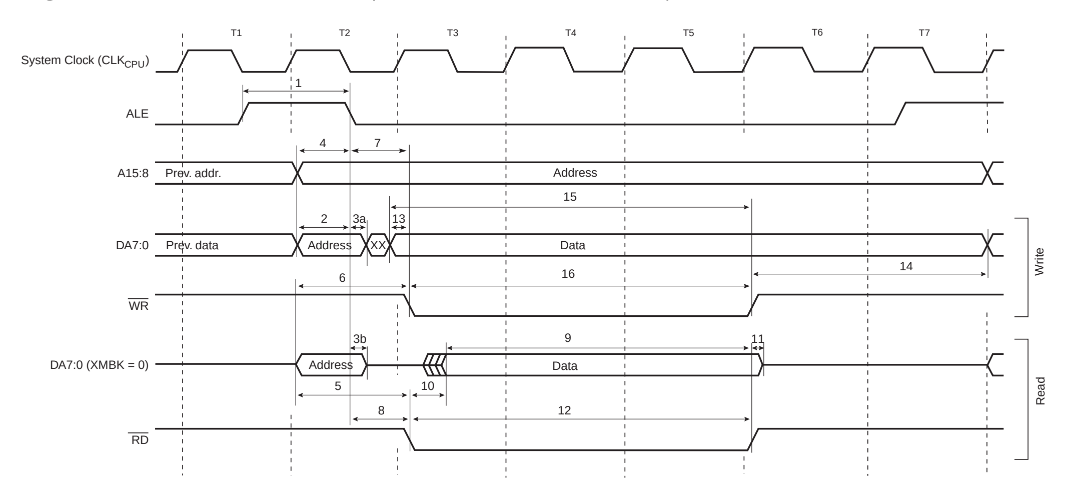


Figure 159. 外部存储器时序 (SRWn1 = 1, SRWn0 = 1)(1)



Note: 1. 最后一个周期 (T4-T7) 的 ALE 脉冲示出下一条指令访问 RAM (内部或外部)。

ATmega128 典型特性

下面图表给出了典型数据。这些数据在产生过程没有进行测试。所有的电流测量数据都是在所有的 I/O 引脚配置为输入且内部上拉电阻使能的条件下测得的。时钟源为外部正弦波发生器产生的满幅值正弦波。

掉电模式下的电流与时钟无关。

电流与多个因素有关，如：工作电压、工作频率、 I/O 引脚的负载及电平转换频率、执行的代码和环境温度。主要因素为工作电压和工作频率。

容性负载引脚的电流可以通过公式 C_L*V_{CC}*f 进行估计。式中，C_L 为负载电容，V_{CC} 为工作电压，f 为引脚的平均开关频率。

器件的特性参数为比测试上限更高的频率下的数据。但是不保证器件在比定货信息表明的工作频率更高的频率功能正常工作。

掉电模式下看门狗使能与看门狗禁止之间的电流差异即是看门狗定时器所需的工作电流。

工作电流

Figure 160. 工作电流与频率 的关系 (0.1 - 1.0 MHz)

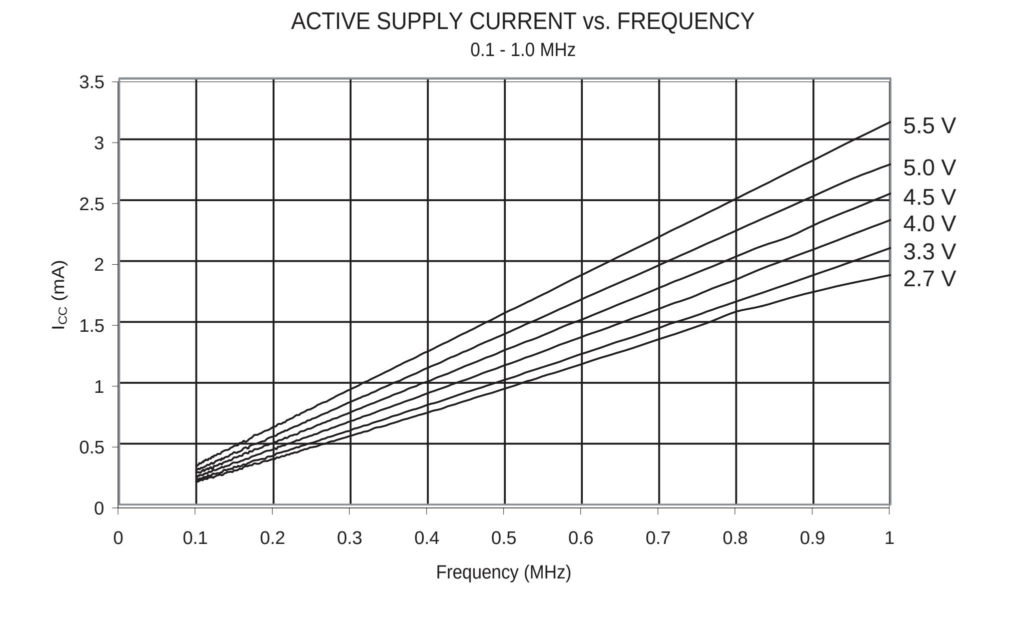






Figure 161. 工作电流与频率 的关系 (1 - 20 MHz)

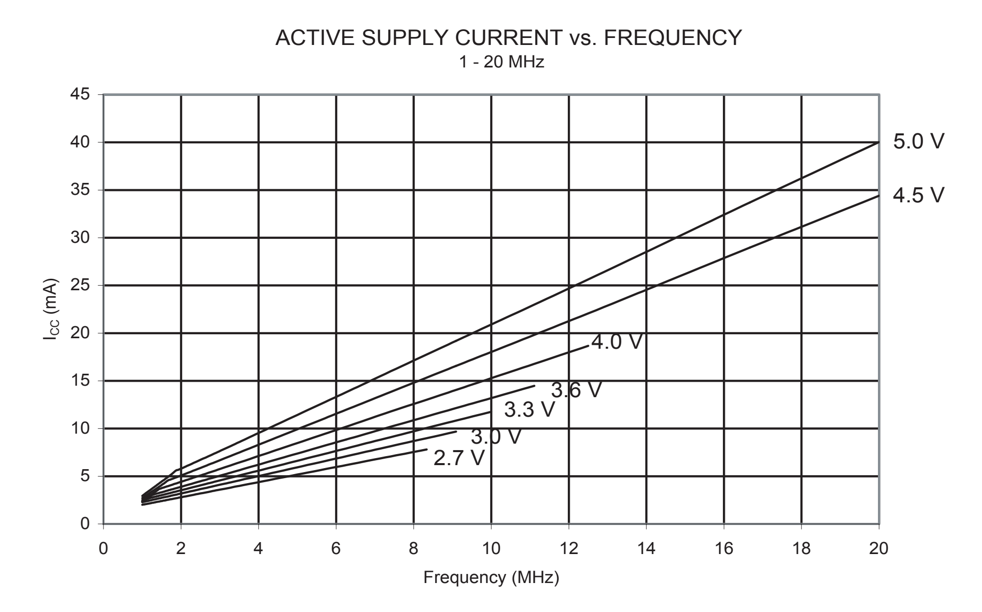


Figure 162. 工作电流与 V_{CC} 的关系 (内部 RC 振荡器，1 MHz)

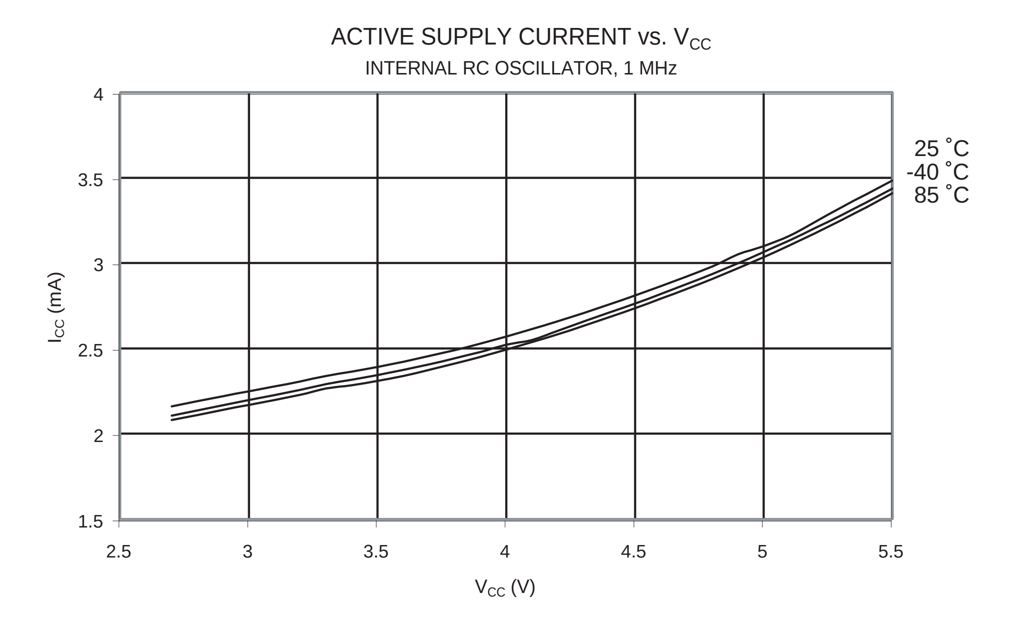


Figure 163. 工作电流与 V_{CC} 的关系 (内部 RC 振荡器 ， 2 MHz)

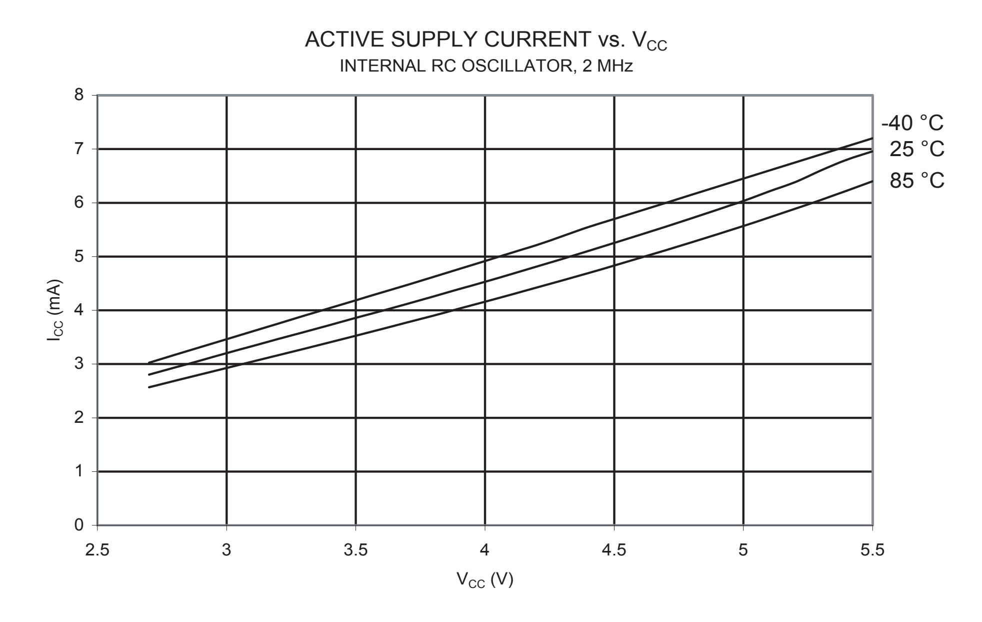


Figure 164. 工作电流与 V_{CC} 的关系 (内部 RC 振荡器 , 4 MHz)

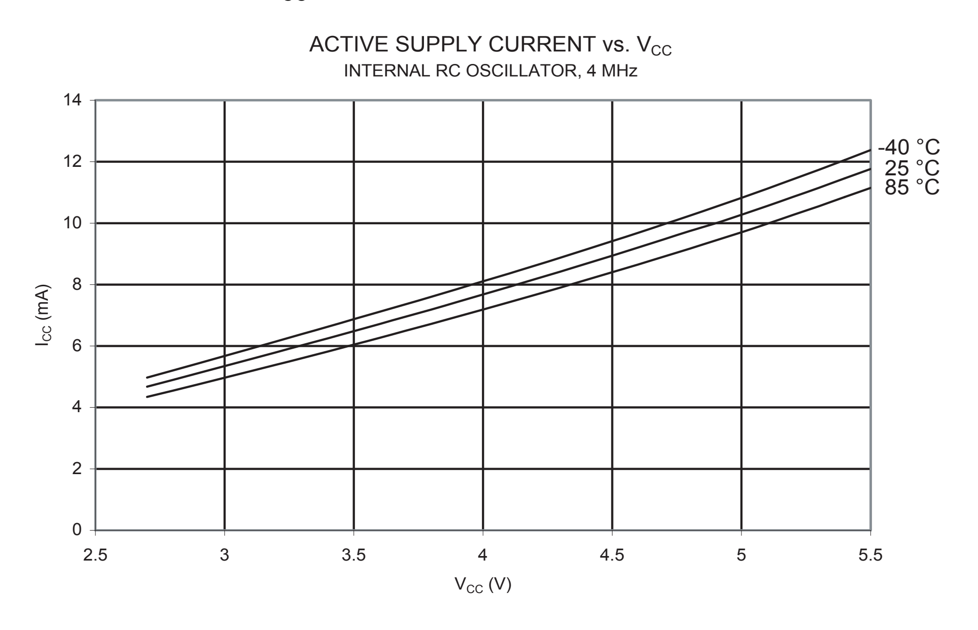




Figure 165. 工作电流与 V_{CC} 的关系 (内部 RC 振荡器，8 MHz)

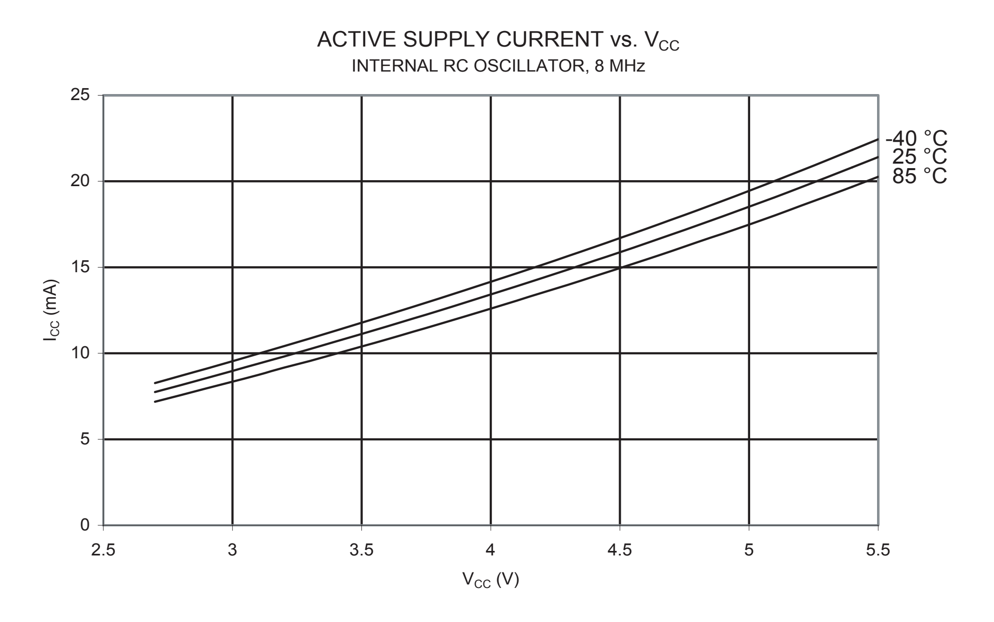
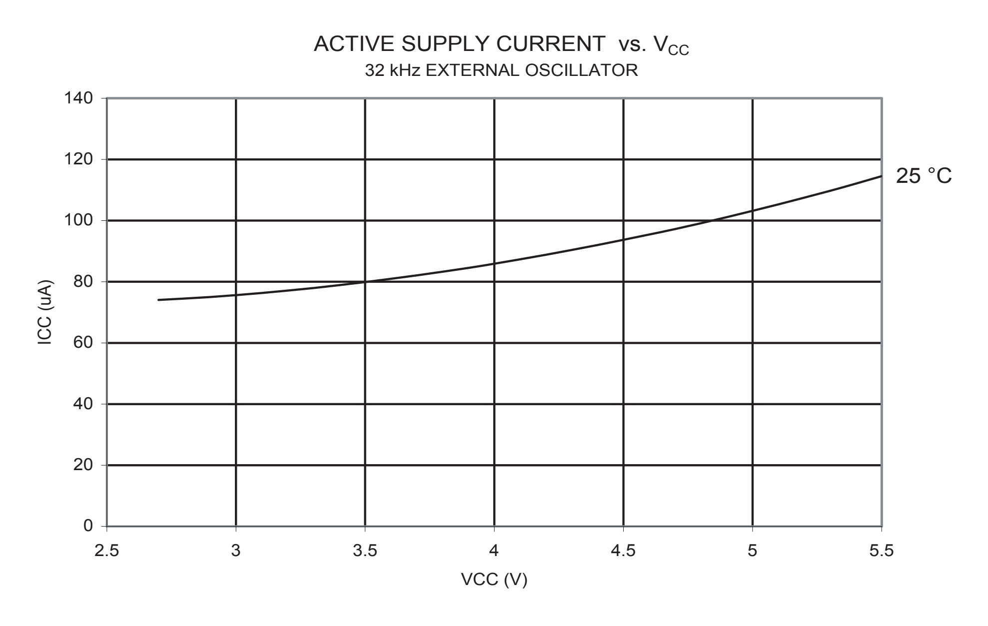


Figure 166. 工作电流与 V_{CC} 的关系 (32 kHz 外部振荡器)



空闲电流

Figure 167. 空闲电流与频率的关系 (0.1 - 1.0 MHz)

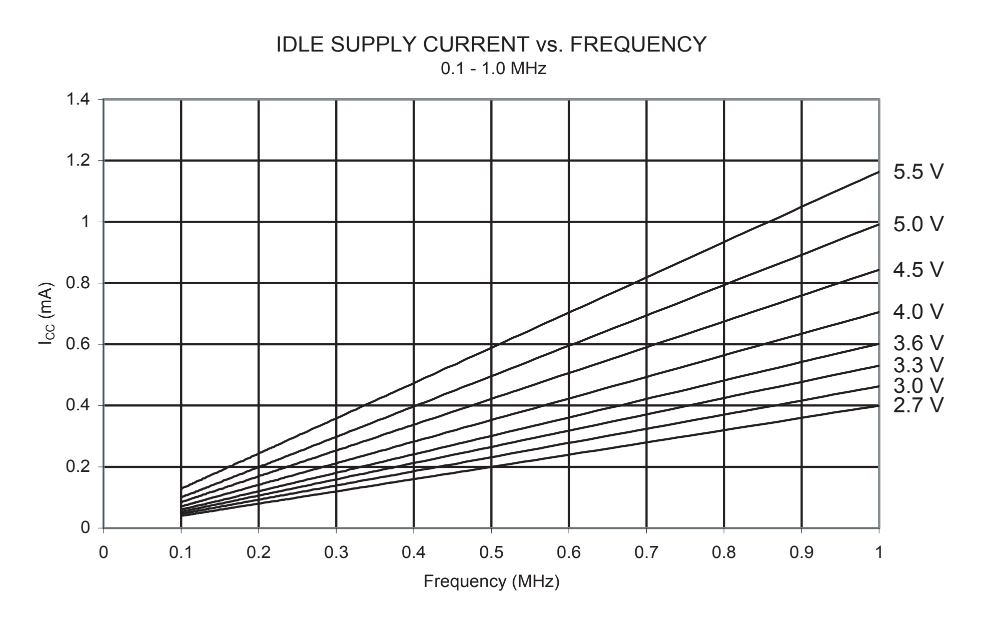


Figure 168. 空闲电流与频率的关系 (1 - 20 MHz)

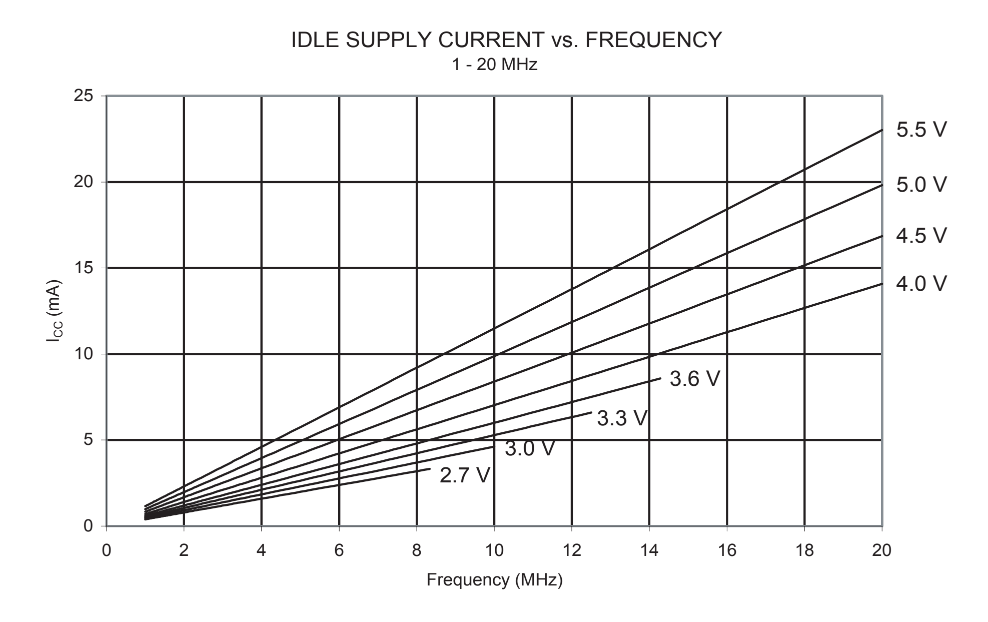


Figure 169. 空闲电流与 V_{CC} 的关系 (内部 RC 振荡器 ， 1 MHz)

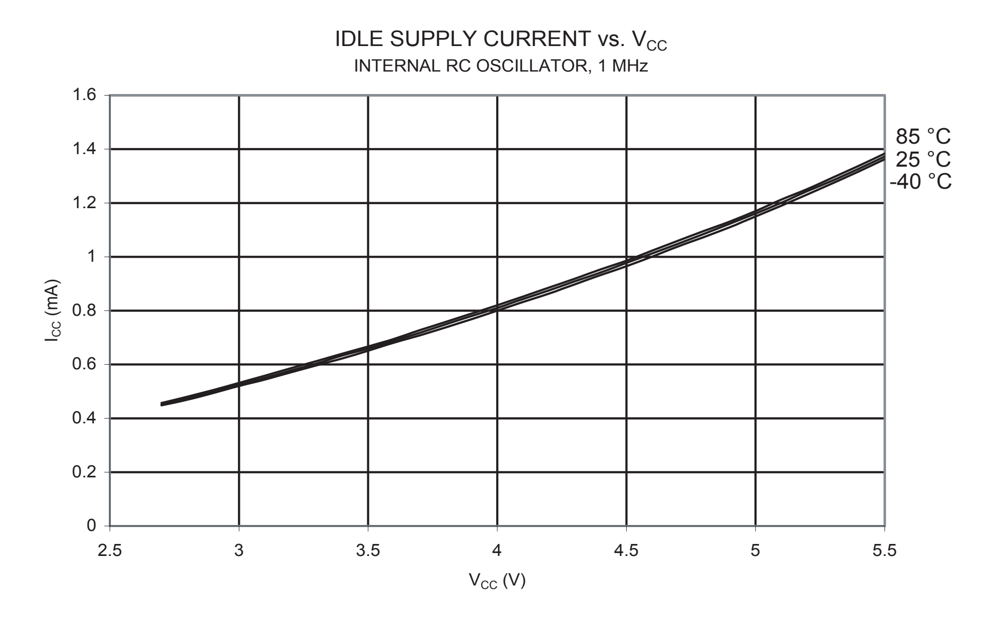


Figure 170. 空闲电流与 V_{CC} 的关系 (内部 RC 振荡器 ，2 MHz)

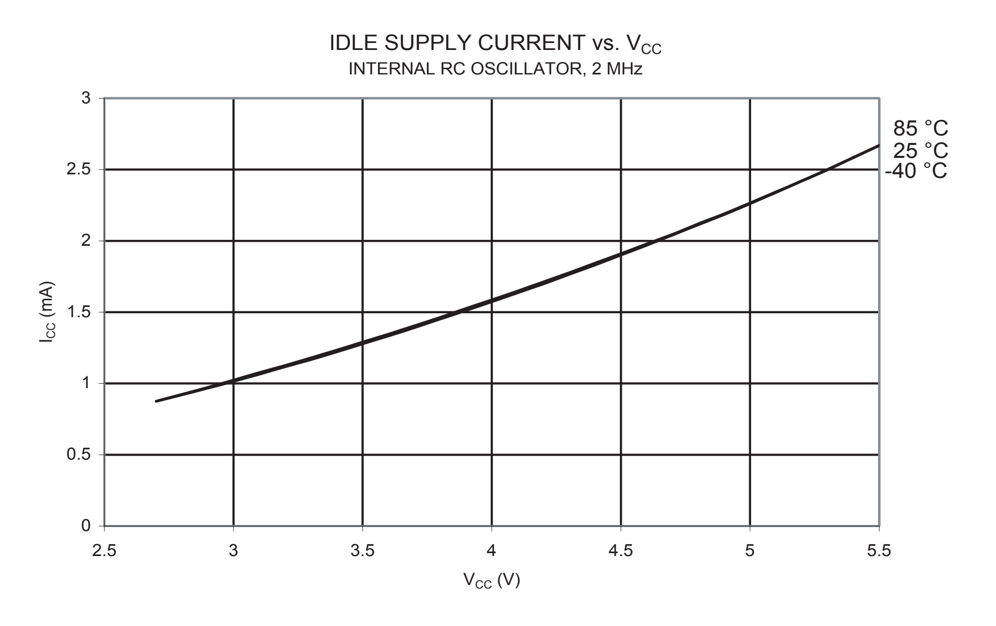


Figure 171. 空闲电流与 V_{CC} 的关系 (内部 RC 振荡器 ， 4 MHz)

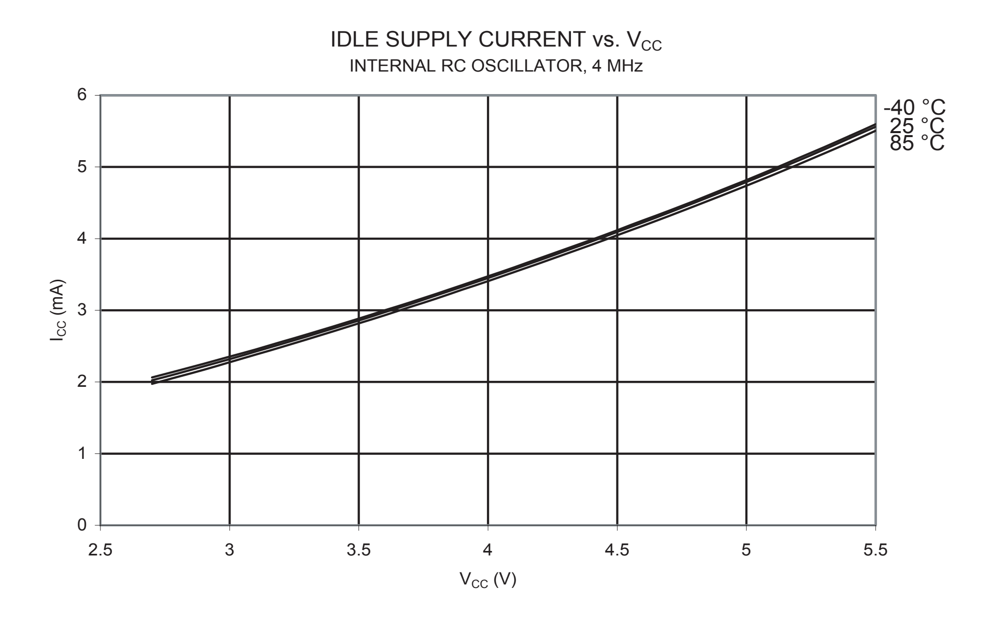


Figure 172. 空闲电流与 V_{CC} 的关系 (内部 RC 振荡器 ，8 MHz)

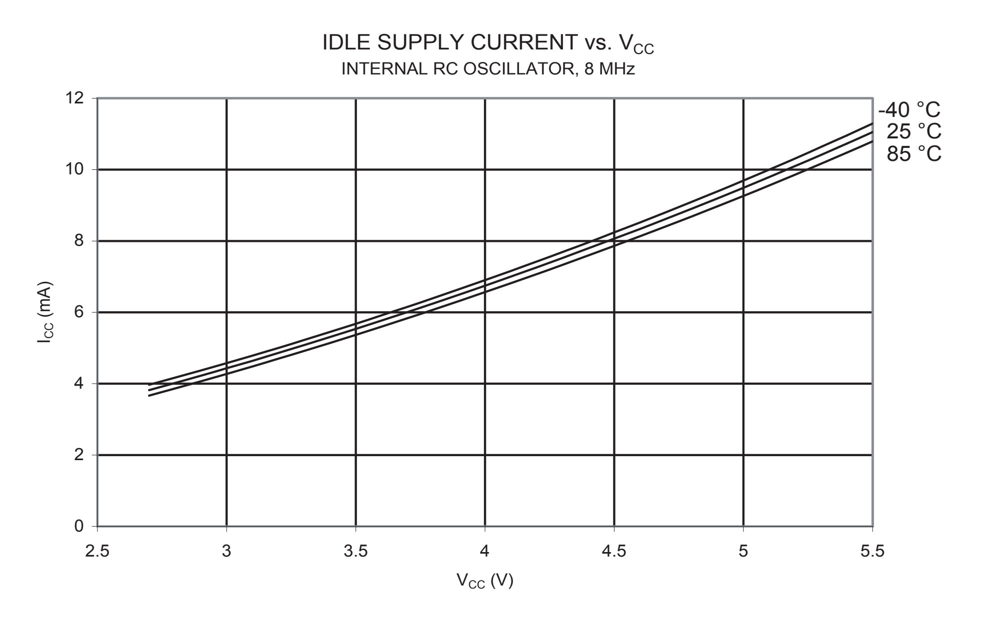
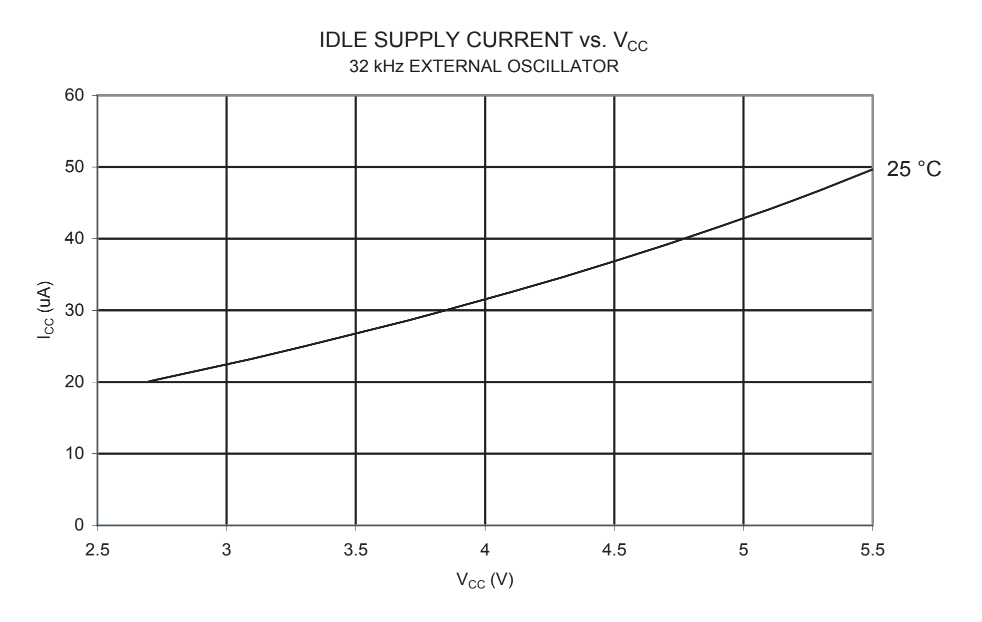


Figure 173. 空闲电流与 V_{CC} 的关系 (32 kHz 外部振荡器)



掉电模式电流

Figure 174. 掉电模式电流与 V_{CC} 的关系 (看门狗定时器禁用)

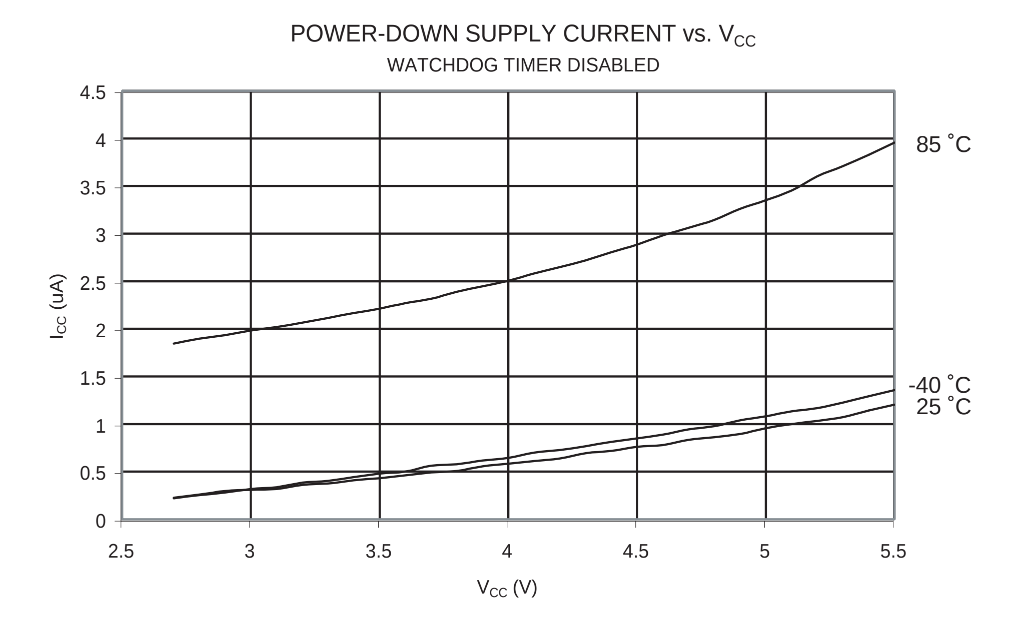
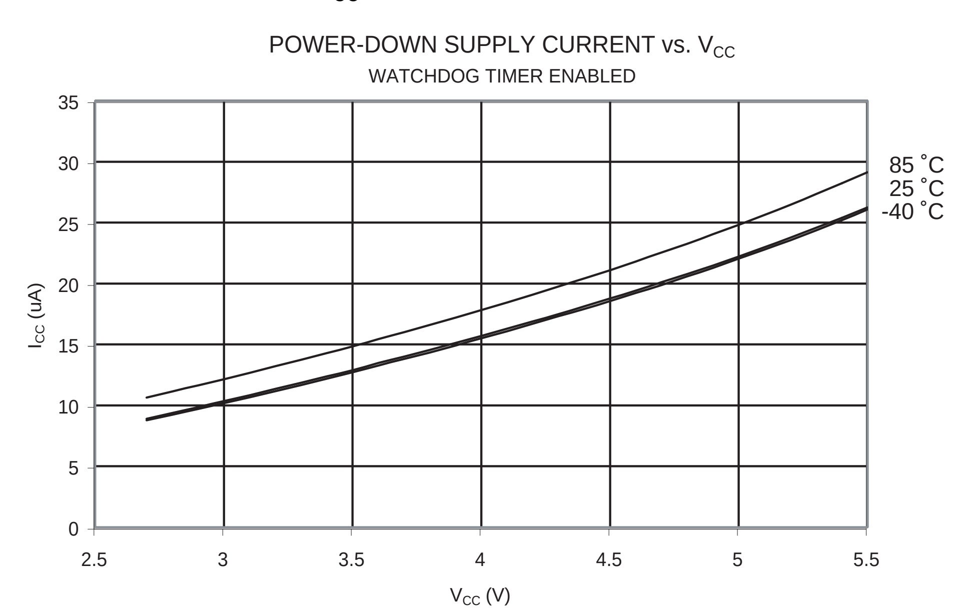
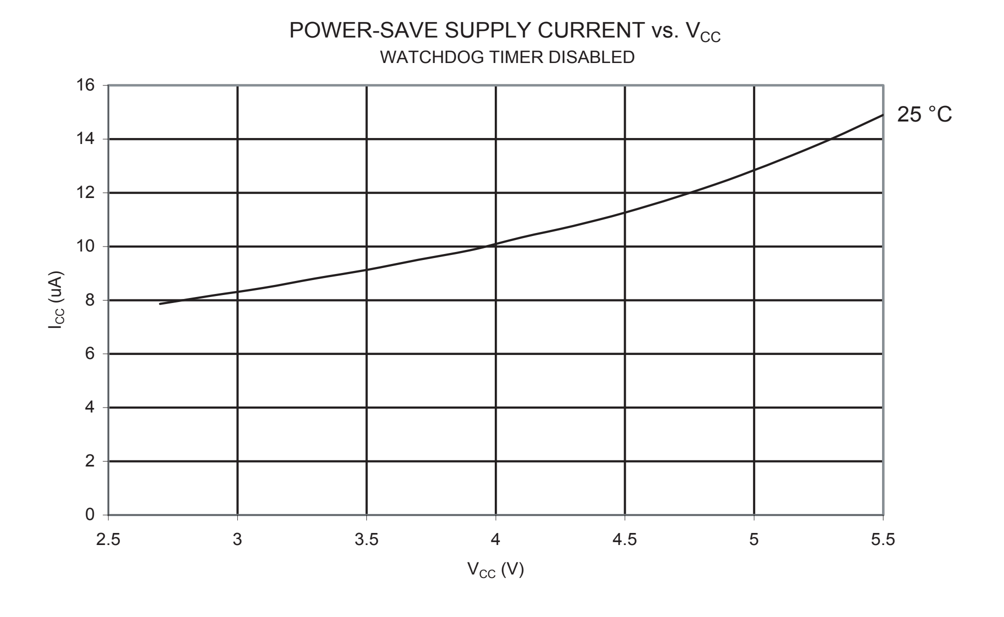


Figure 175. 掉电模式电流与 V_{CC} 的关系 (看门狗定时器使能)



省电模式电流

Figure 176. 省电模式电流与 V_{CC} 的关系 (看门狗定时器禁用)



Standby 模式电流

Figure 177. Standby 模式电流与 V_{CC}, 的关系

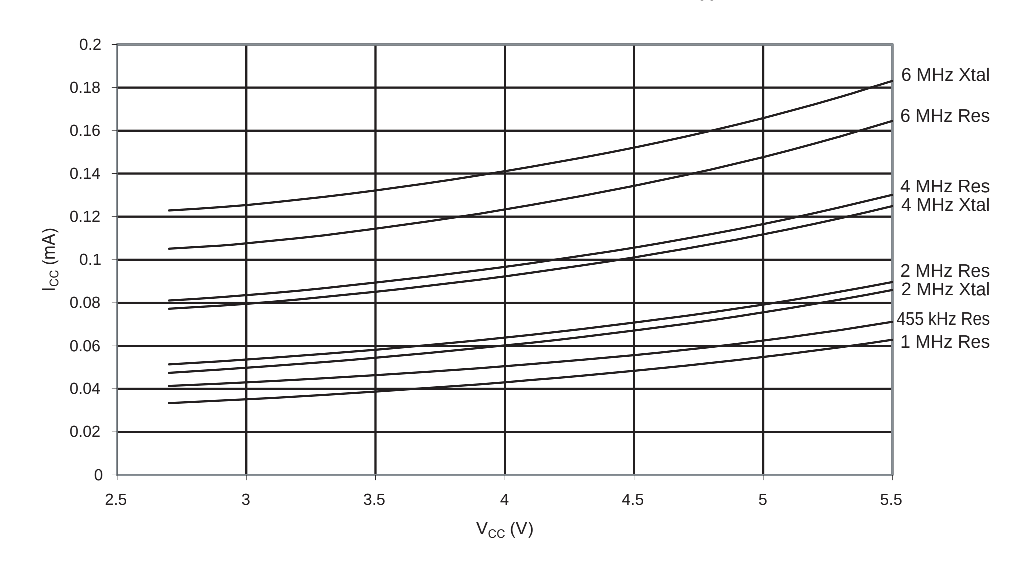
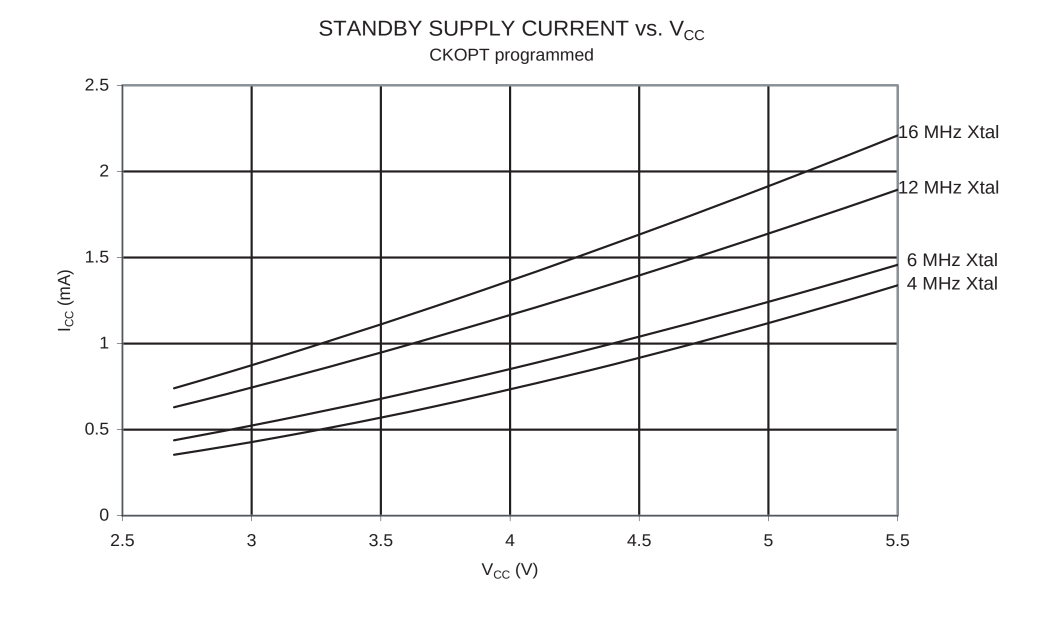


Figure 178. Standby 模式电流与 V_{CC}, 的关系 (CKOPT 编程)



引脚上拉电阻

Figure 179. I/O 引脚上拉电阻电流与输入电压的关系 (V_{CC} = 5V)

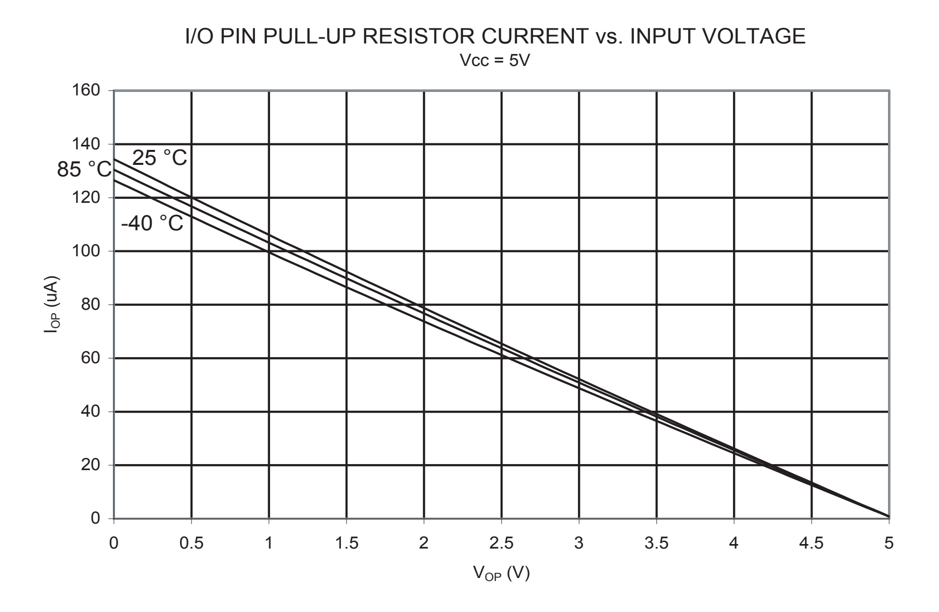
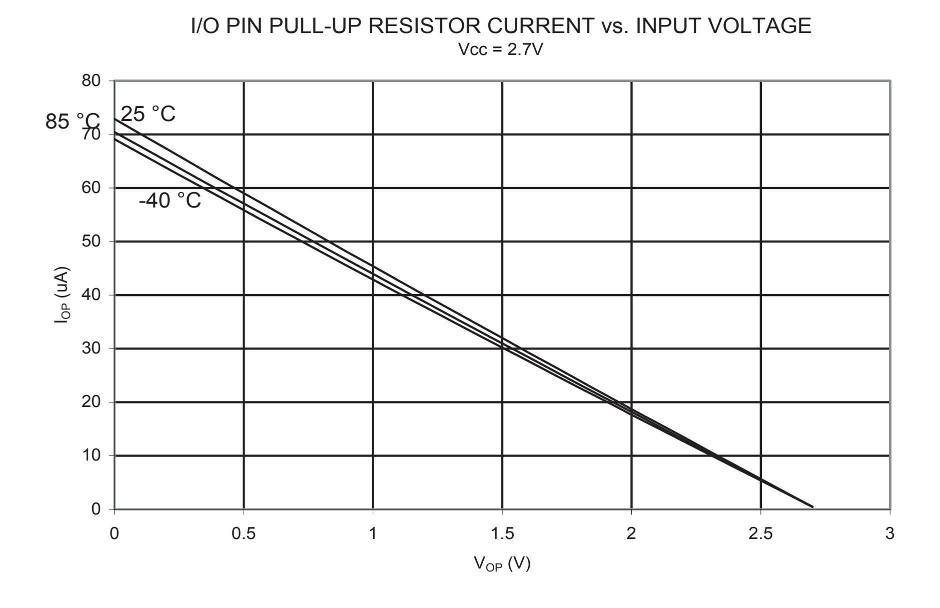


Figure 180. I/O 引脚上拉电阻电流与输入电压的关系 (V_{CC} = 2.7V)



引脚驱动能力

Figure 181. I/O 引脚源电流与输出电压的关系 (V_{CC} = 5V)

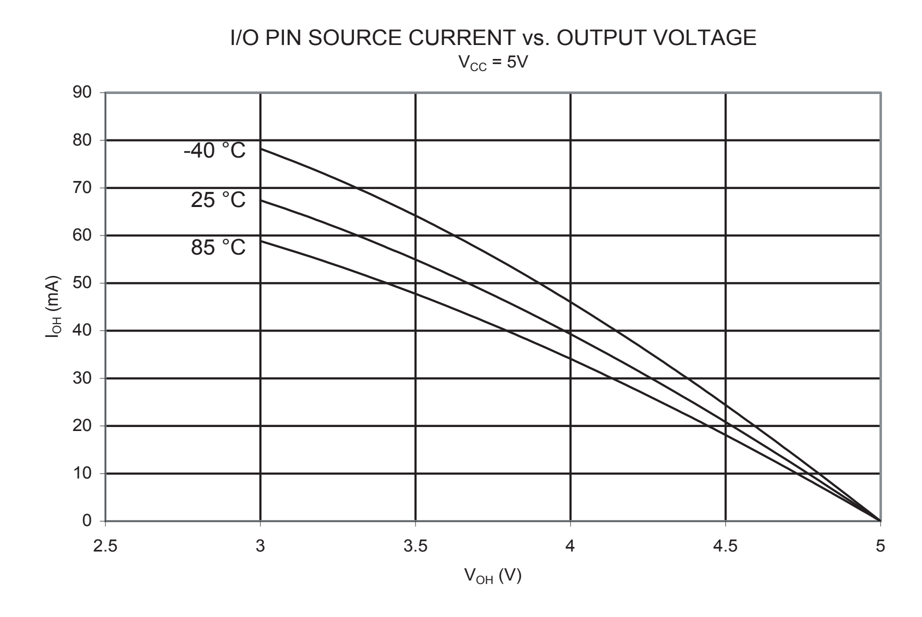


Figure 182. I/O 引脚源电流与输出电压的关系 (V_{CC} = 2.7V)

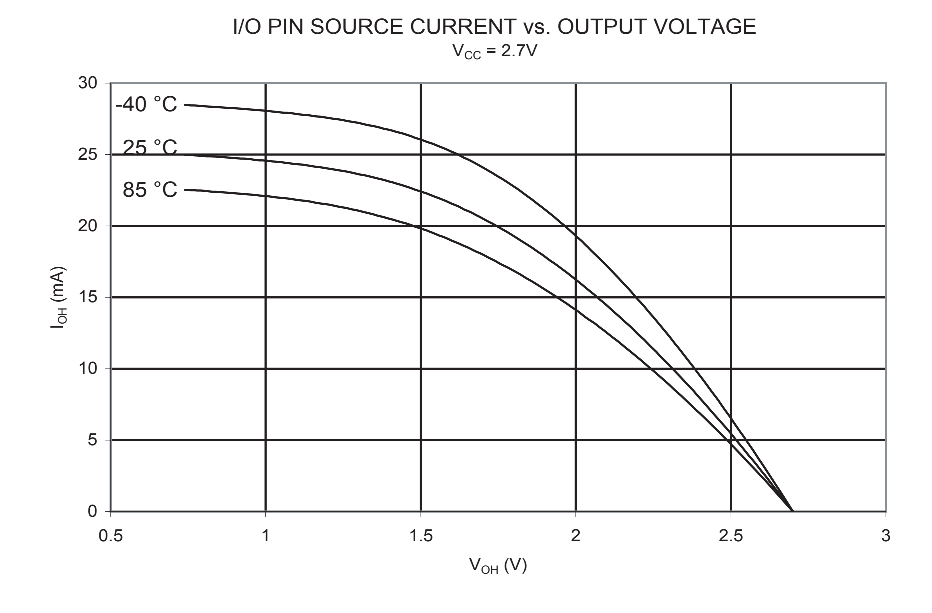


Figure 183. I/O 引脚漏电流与输出电压的关系 (V_{CC} = 5V)

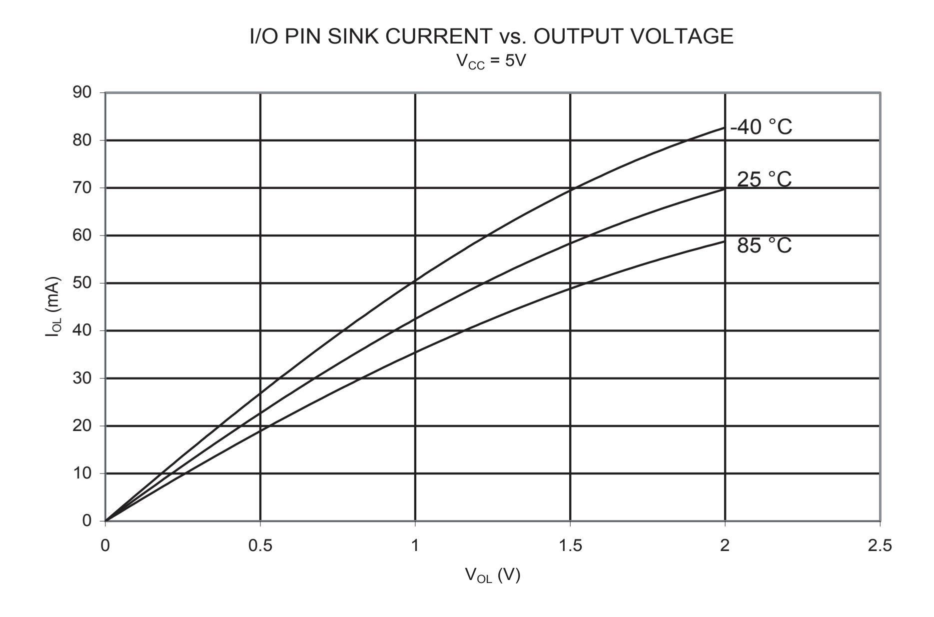
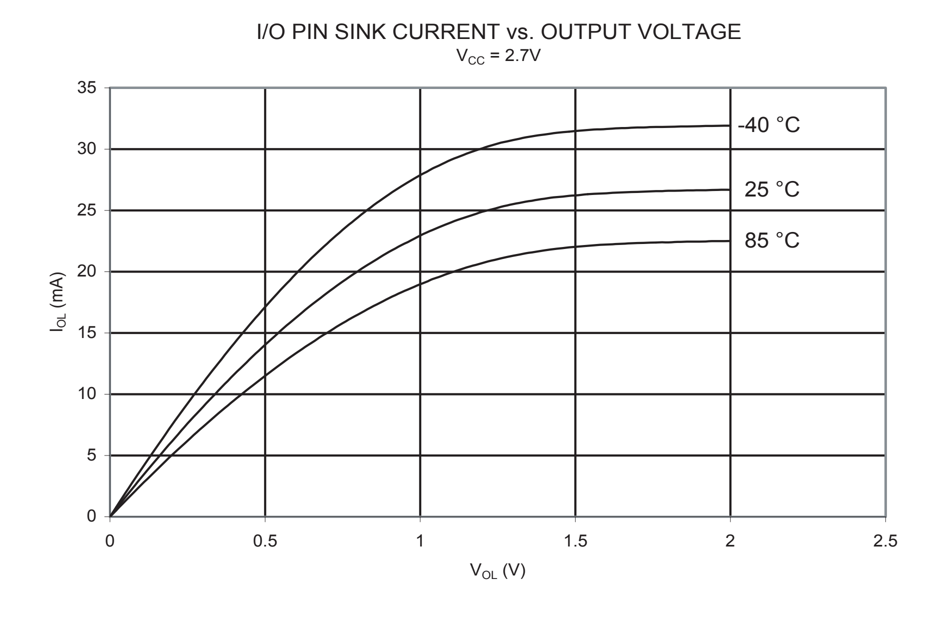


Figure 184. I/O 引脚漏电流与输出电压的关系，V_{CC} = 2.7V



引脚阈值与迟滞

Figure 185. I/O 引脚输入阈值电压与 V_{CC} 的关系 (VIH， I/O 引脚读为 ‘1’)

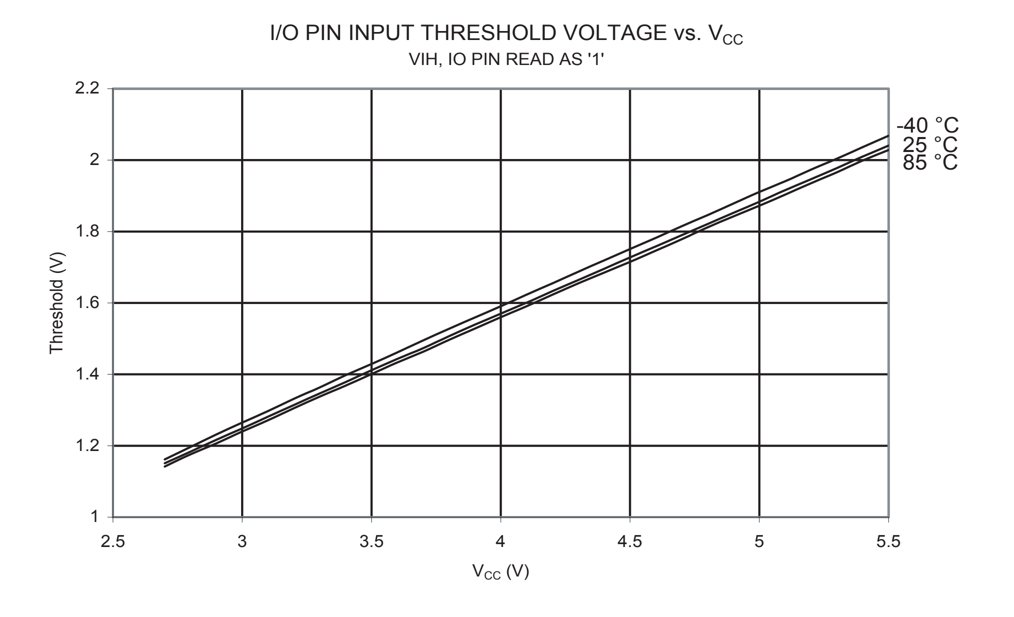


Figure 186. I/O 引脚输入阈值电压与 V_{CC} 的关系 (VIH， I/O 引脚读为 ‘0’)

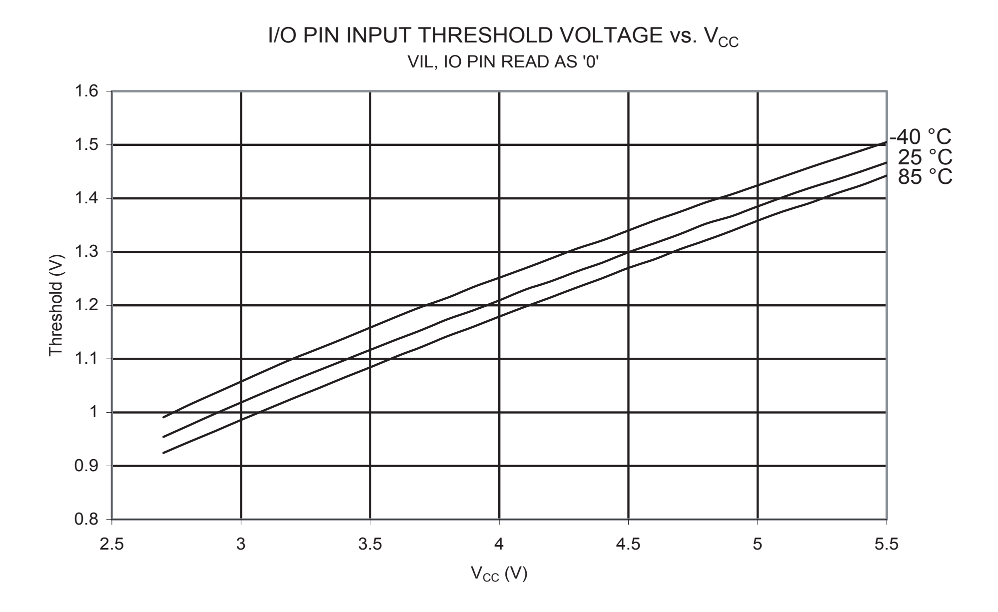
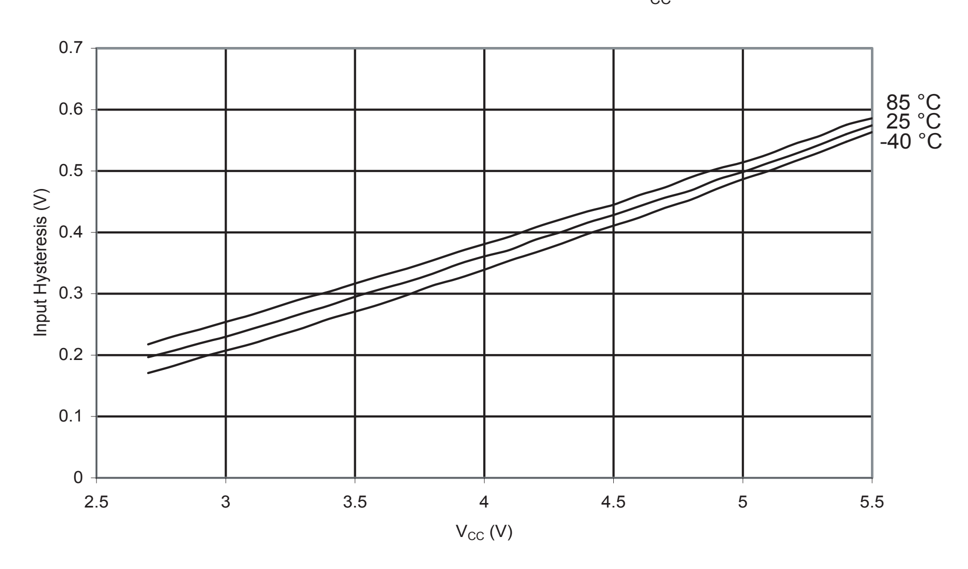


Figure 187. I/O 引脚输入迟滞与 V_{CC} 的关系



BOD 阈值与模拟比较补偿

Figure 188. BOD 阈值与温度的关系 (BODLEVEL 为 4.0V)

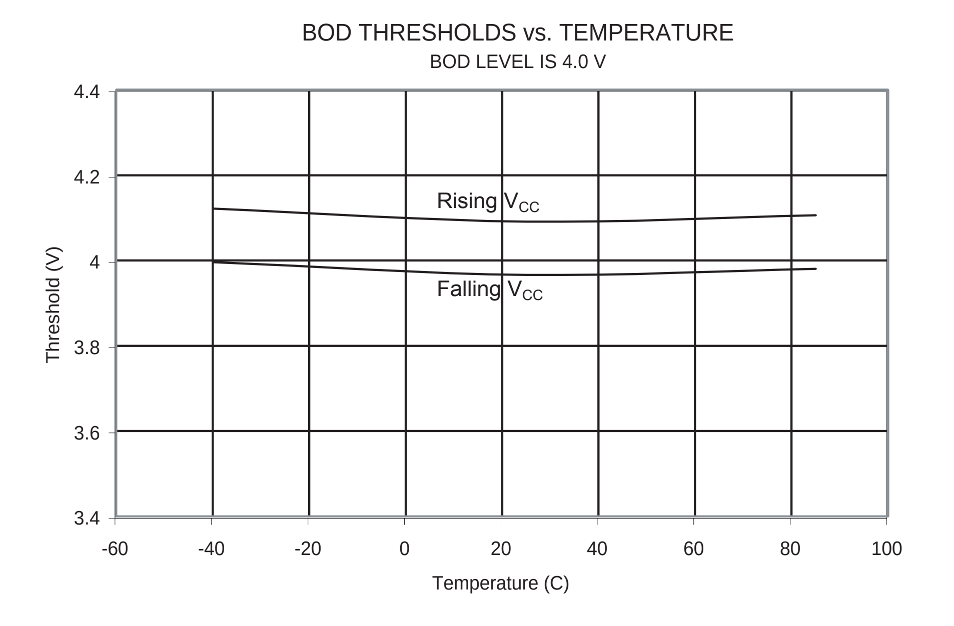


Figure 189. BOD 阈值与温度的关系 (BODLEVEL 为 2.7V)

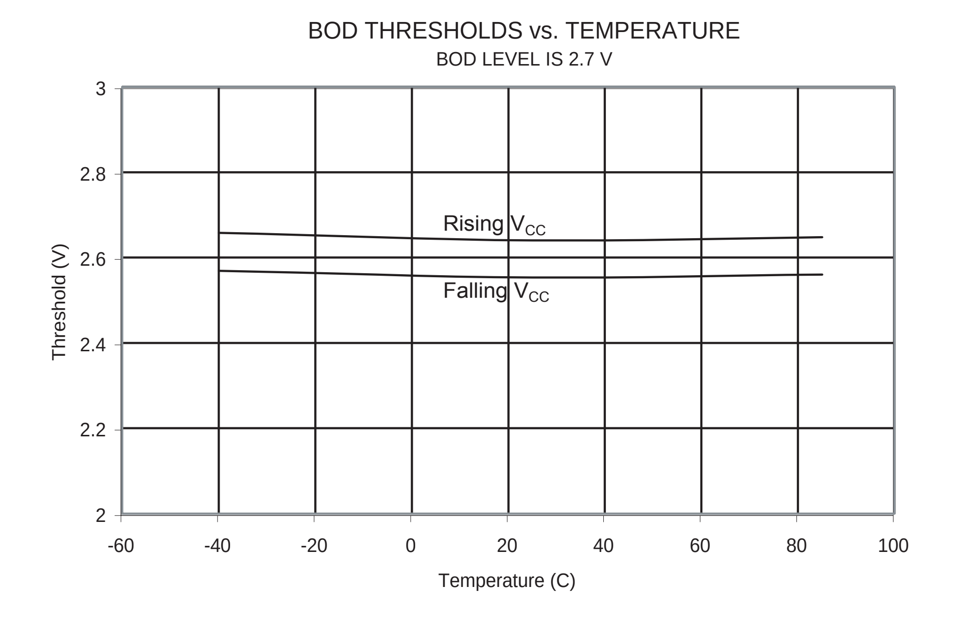
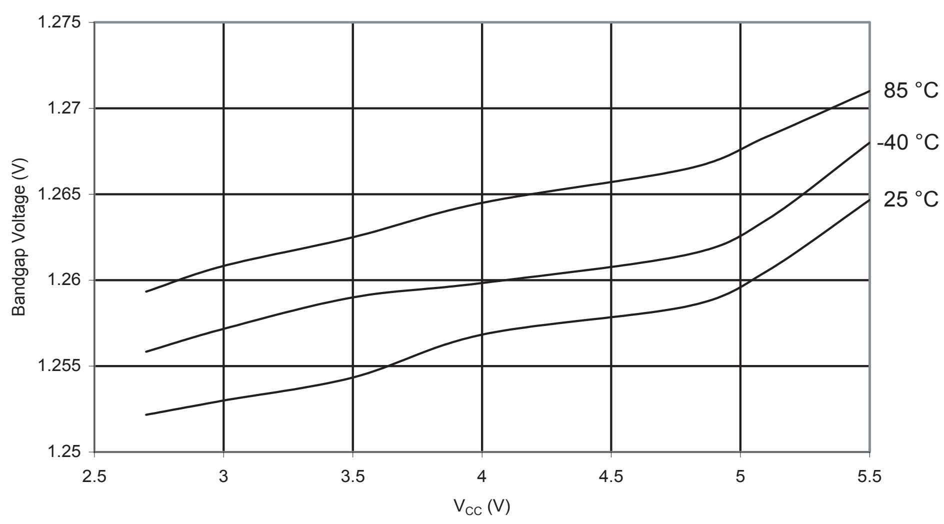


Figure 190. 能隙电压与工作电压的关系





片内振荡器速度

Figure 191. 看门狗振荡器频率与 V_{CC} 的关系

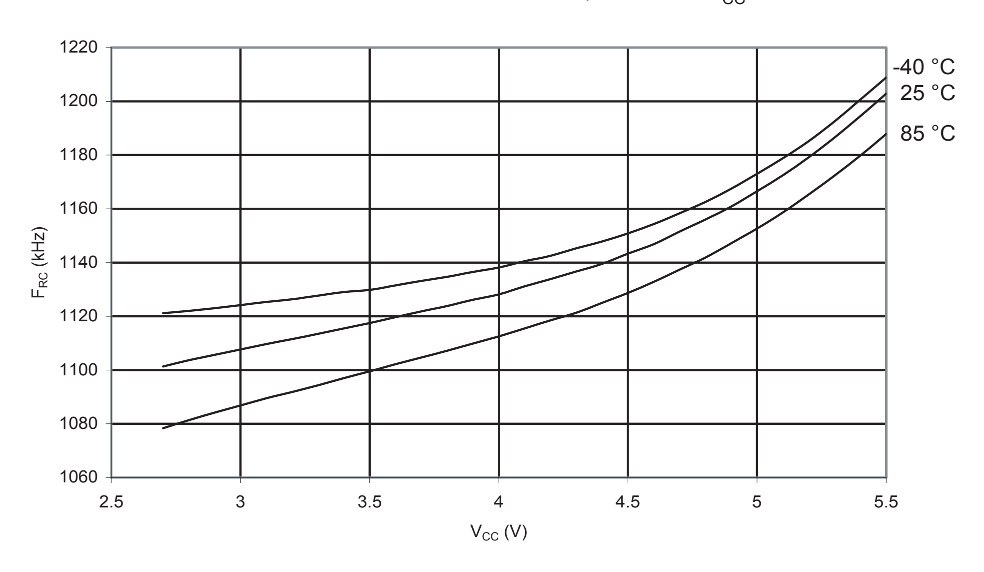


Figure 192. 标定为 1 MHz 的 RC 振荡器频率与温度的关系

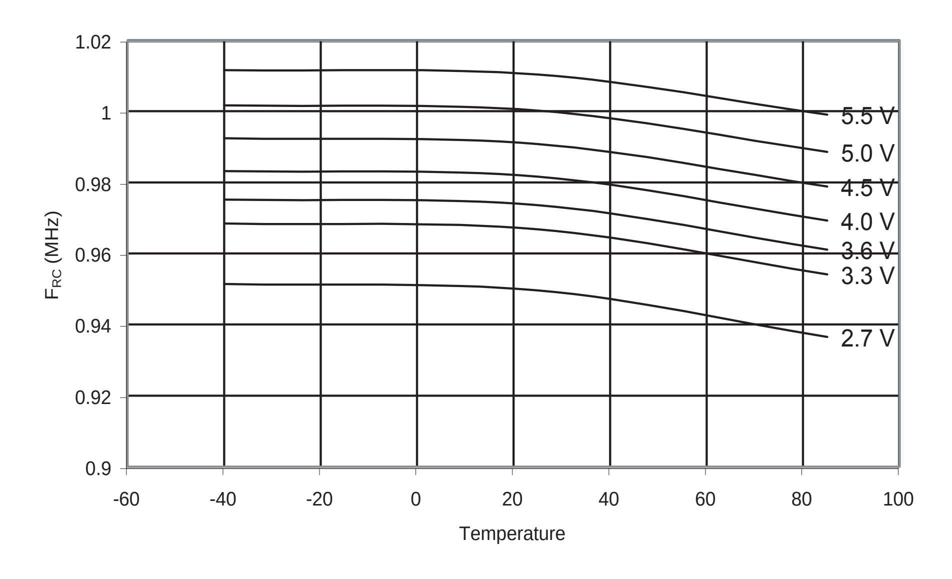




Figure 193. 标定为 1 MHz 的 RC 振荡器频率与 V_{CC} 的关系

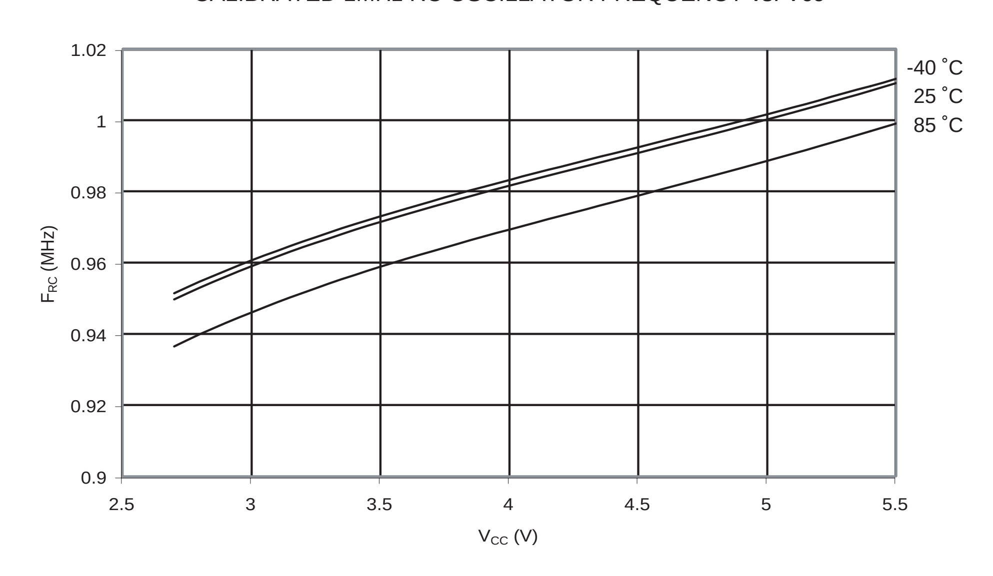


Figure 194. 1 MHz RC 振荡器频率与 Osccal 值的关系

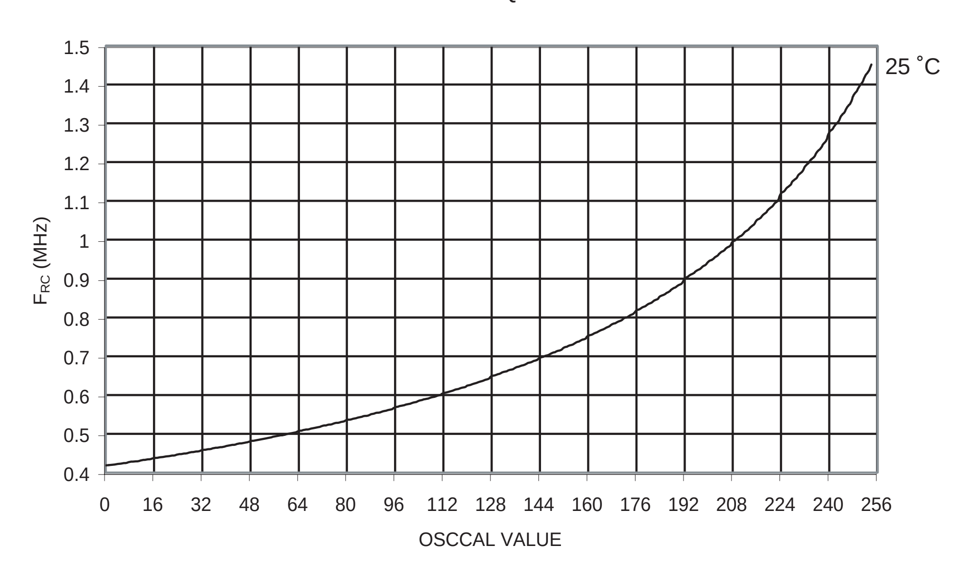


Figure 195. 标定为 2 MHz 的 RC 振荡器频率与温度的关系

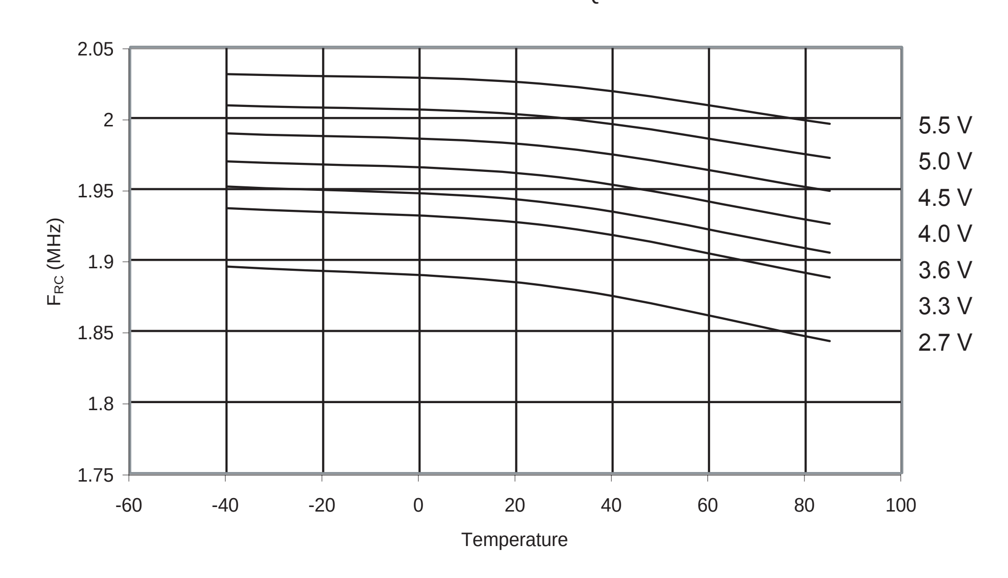


Figure 196. 标定为 2 MHz 的 RC 振荡器频率与 V_{CC} 的关系

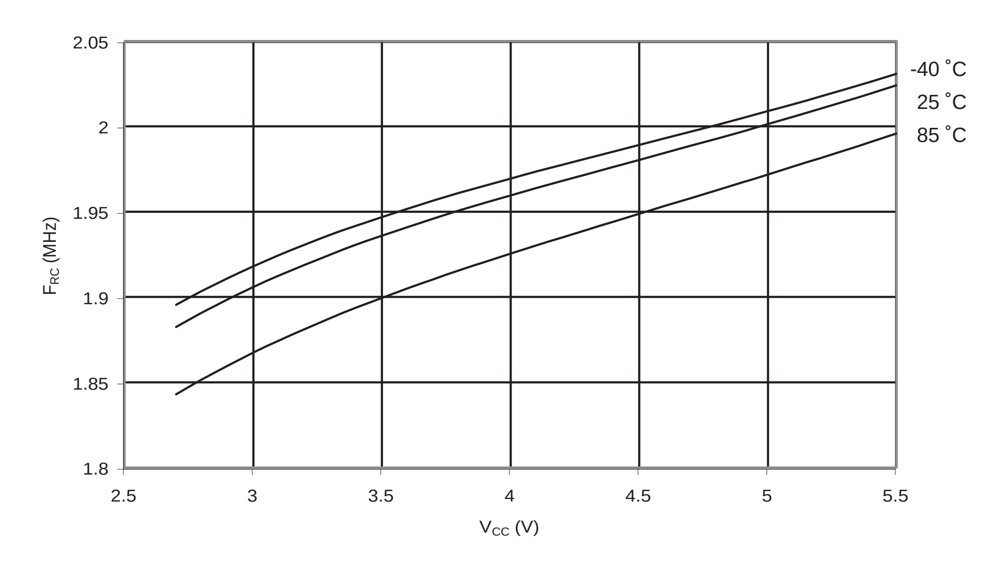


Figure 197. 2 MHz RC 振荡器频率与 Osccal 值的关系

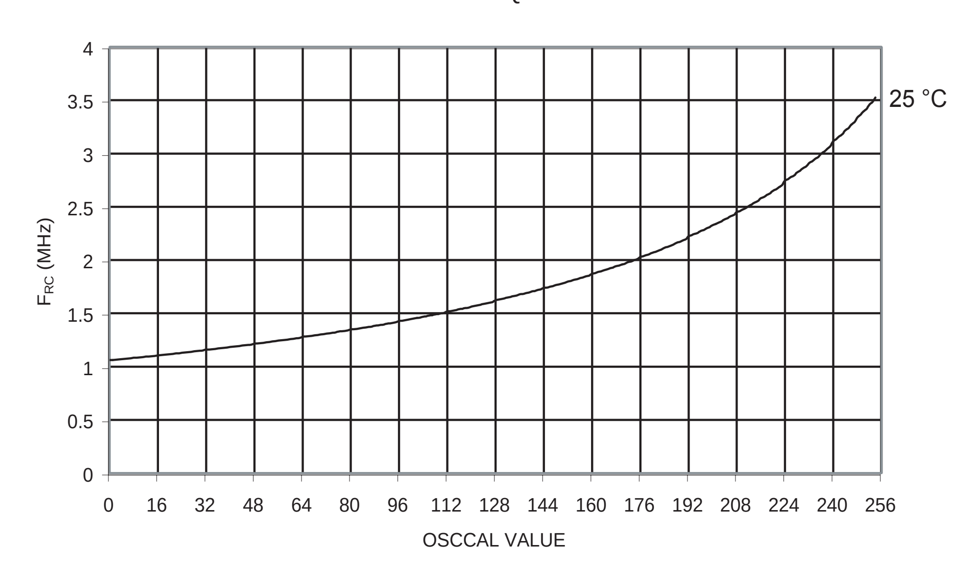


Figure 198. 标定为 4 MHz 的 RC 振荡器频率与温度的关系

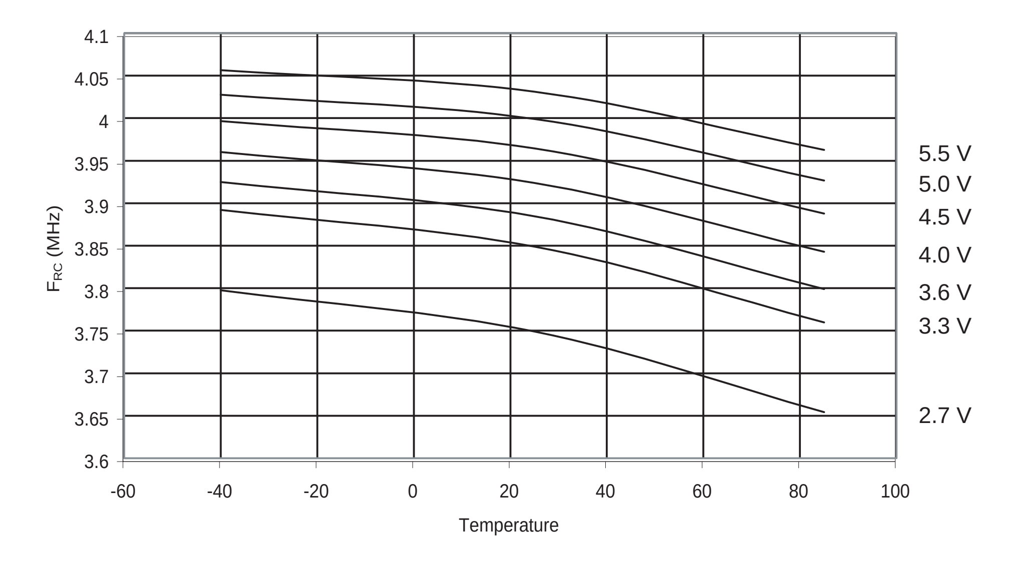


Figure 199. 标定为 4 MHz 的 RC 振荡器频率与 V_{CC} 的关系

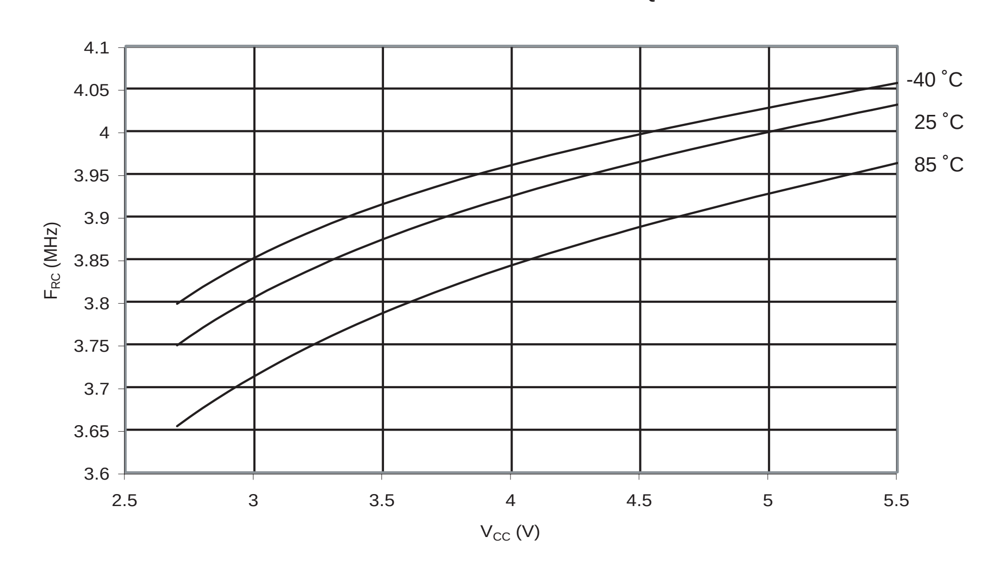


Figure 200. 4 MHz RC 振荡器频率与 Osccal 值的关系

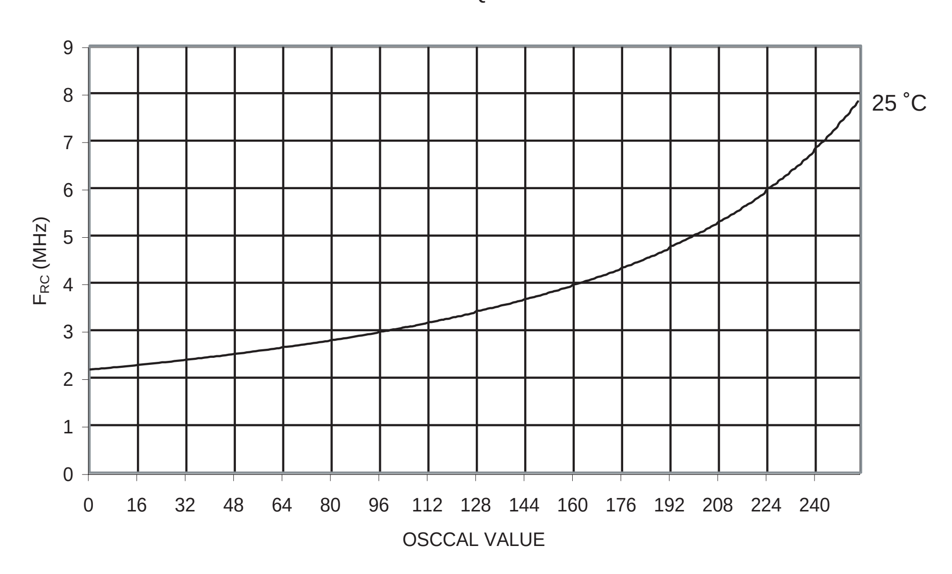


Figure 201. 标定为 8 MHz 的 RC 振荡器频率与温度的关系

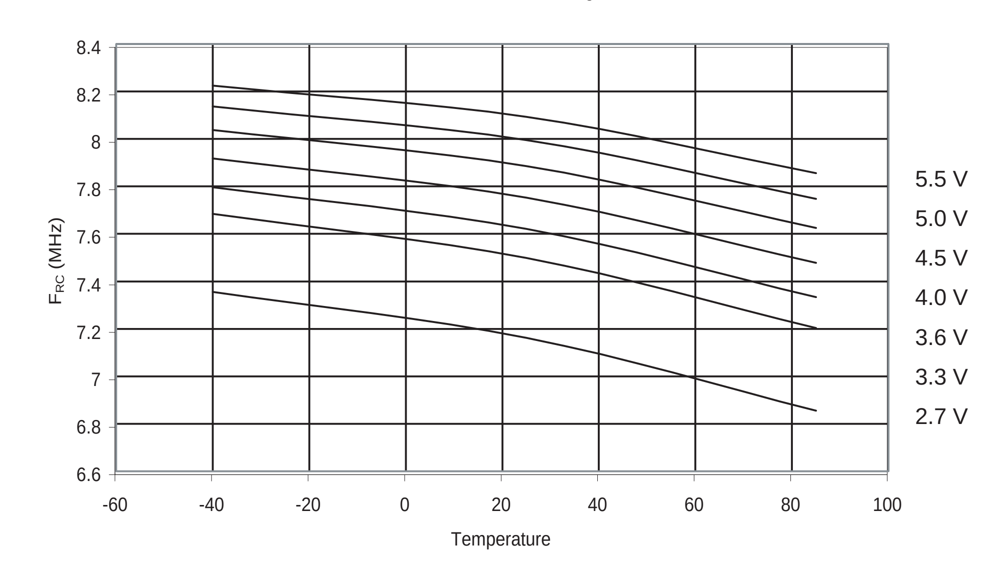


Figure 202. 标定为 8 MHz 的 RC 振荡器频率与 V_{CC} 的关系

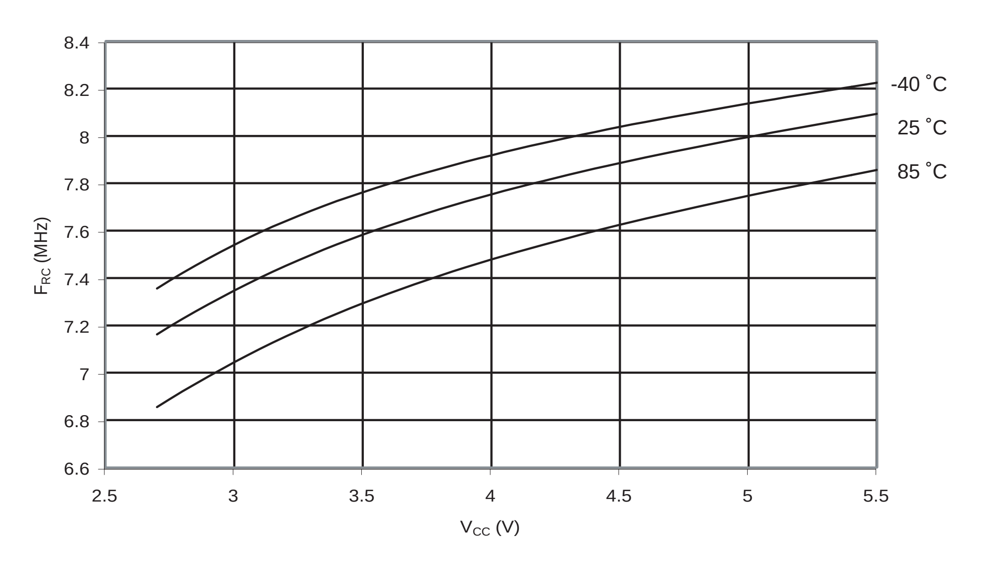
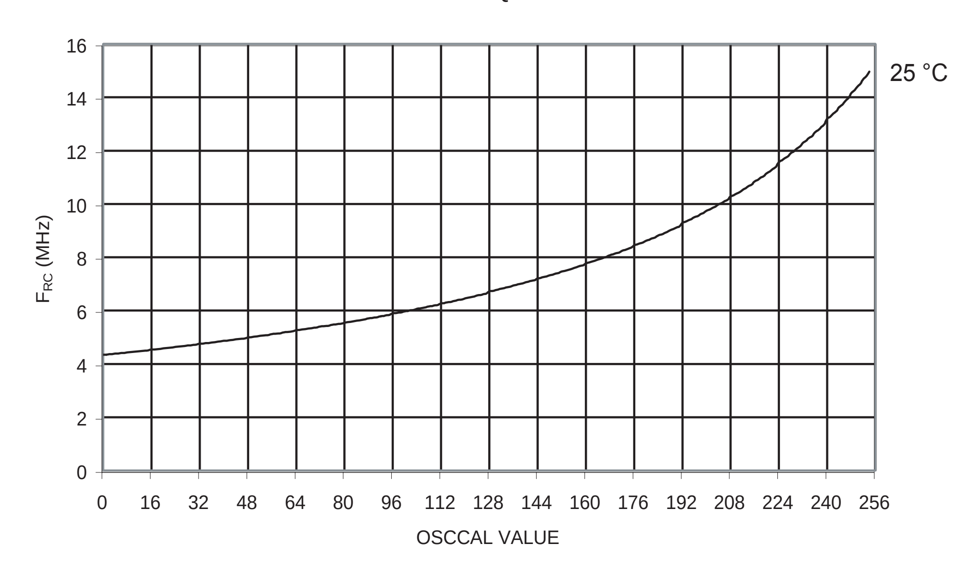


Figure 203. 8 MHz RC 振荡器频率与 Osccal 值的关系



外设单元电流消耗

Figure 204. BOD 电流与 V_{CC} 的关系

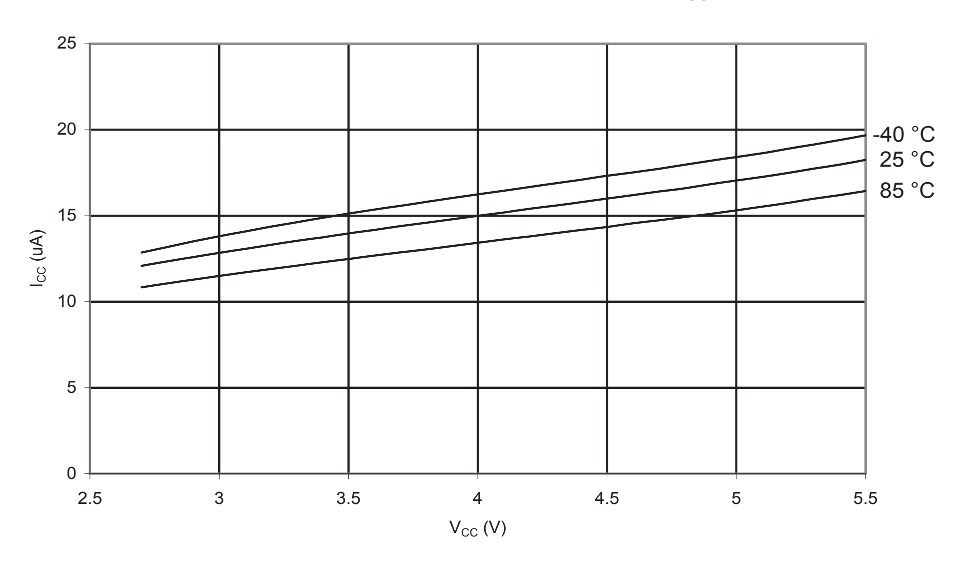




Figure 205. ADC 电流与 V_{CC} 的关系 (ADC 频率为 50 kHz

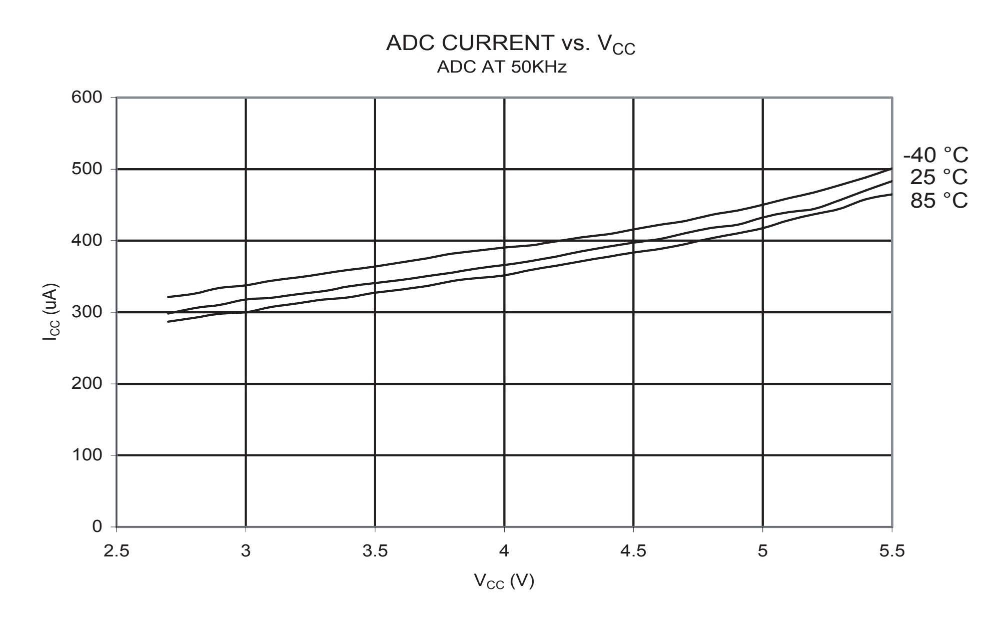


Figure 206. ADC 电流与 V_{CC} 的关系 (ADC 频率为 1 MHz)

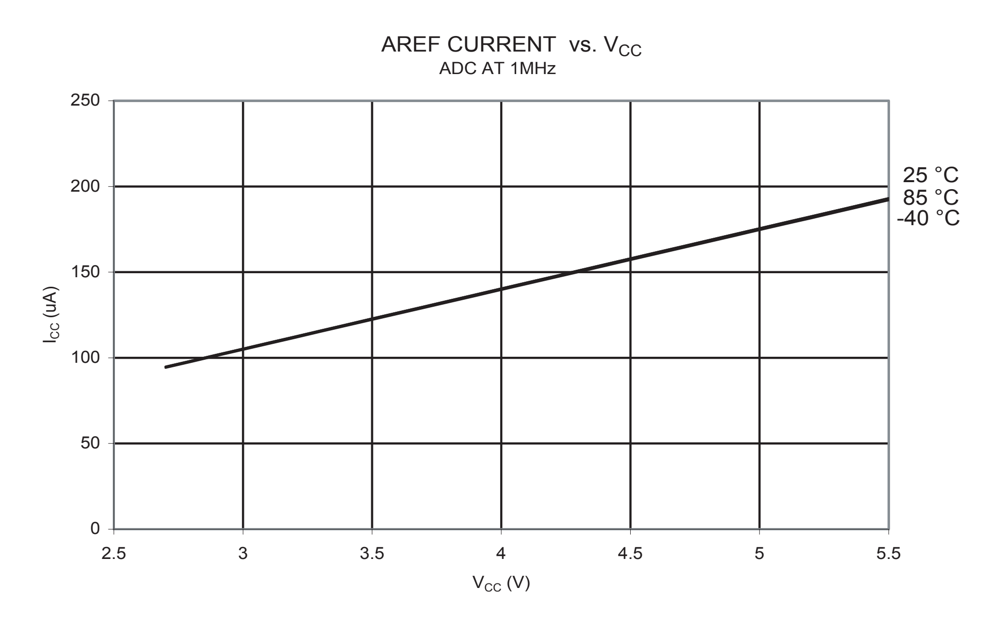


Figure 207. 模拟比较器电流与 V_{CC} 的关系

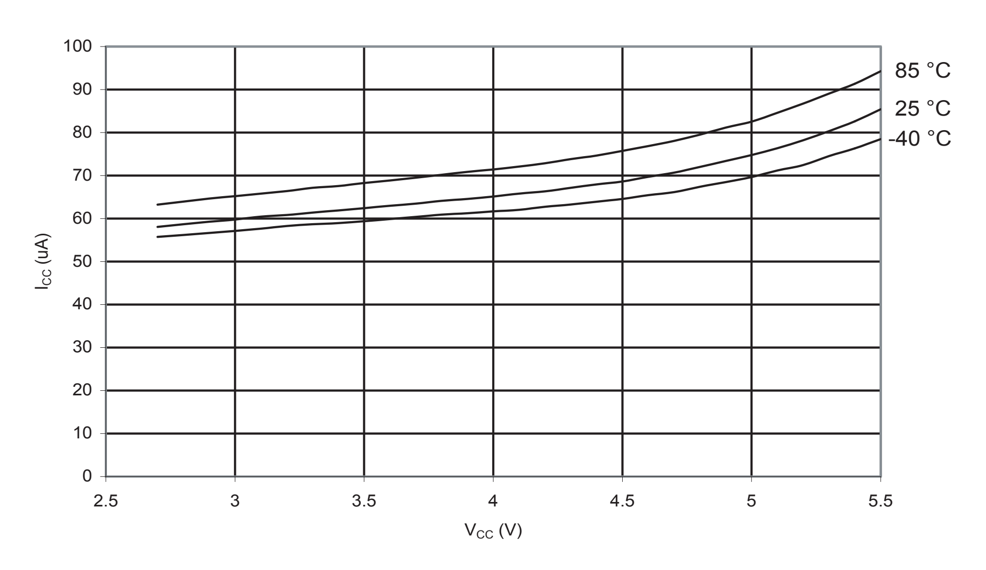
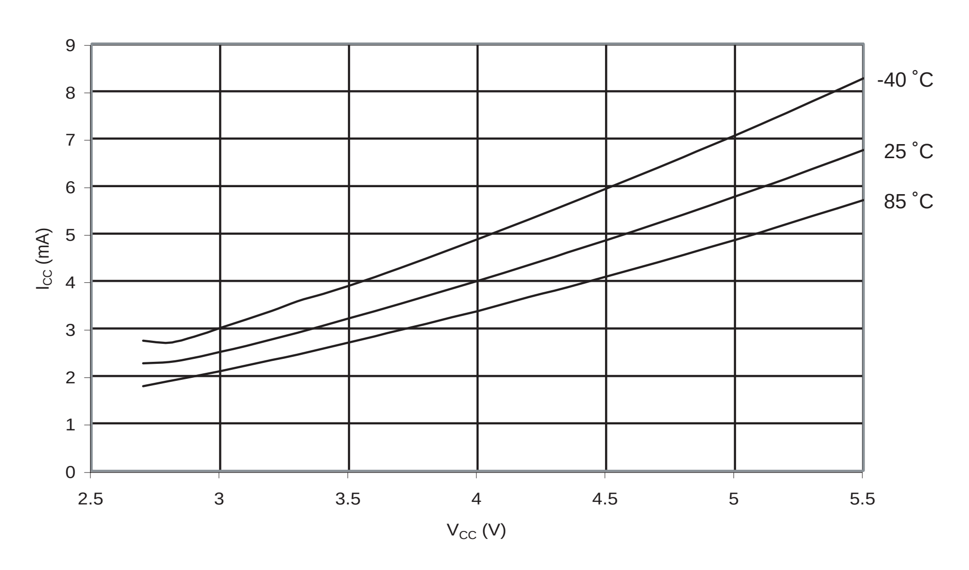


Figure 208. 编程电流与 V_{CC} 的关系



复位与复位脉宽电流消耗

Figure 209. 复位工作电流与 V_{CC} 的关系 (0.1 - 1.0 MHz，包括复位上拉电阻电流)

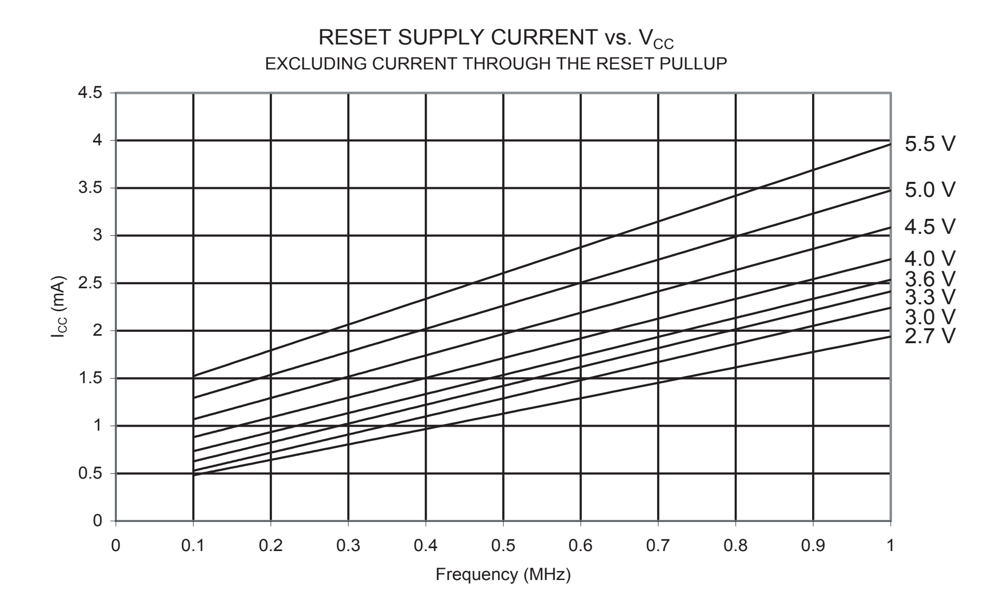


Figure 210. 复位工作电流与 V_{CC} 的关系 (1 - 20MHz，包括复位上拉电阻电流)

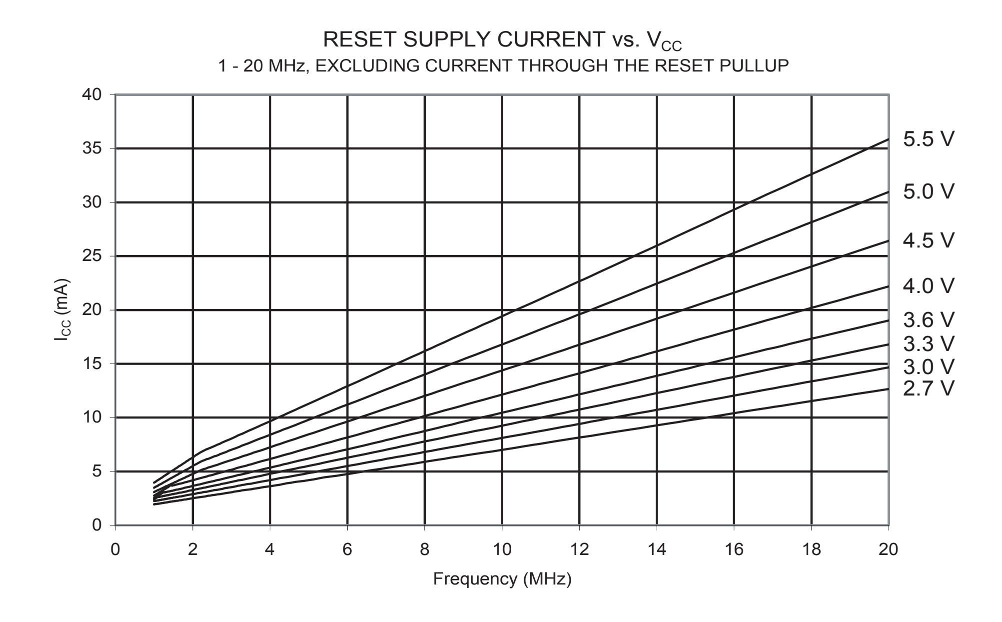


Figure 211. 复位上拉电阻电流与复位引脚电压的关系 (V_{CC} = 5.0V)

RESET PULL-UP RESISTOR CURRENT vs. RESET PIN VOLTAGE

V_{CC} = 5V

-40 °C, 25 °C, 85 °C

I_{RESET} (uA)

V_{RESET} (V)

Figure 212. 复位上拉电阻电流与复位引脚电压的关系 (V_{CC} = 2.7V

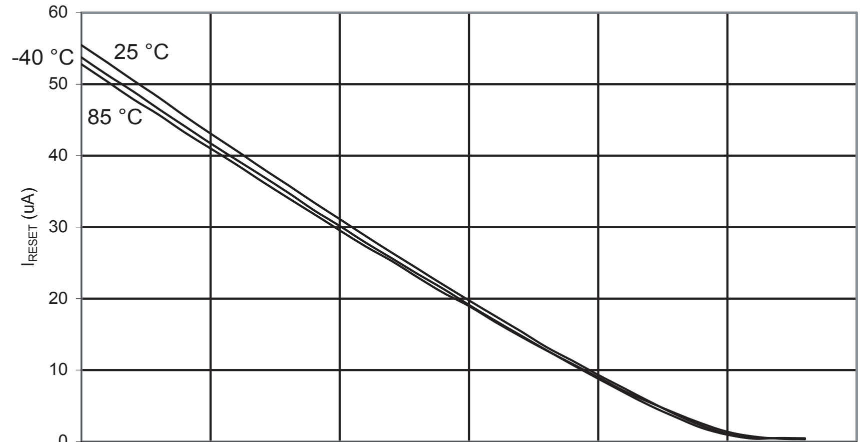


Figure 213. 复位输入阈值电压与 V_{CC} 的关系 (VIH，复位引脚读为 ‘1’)

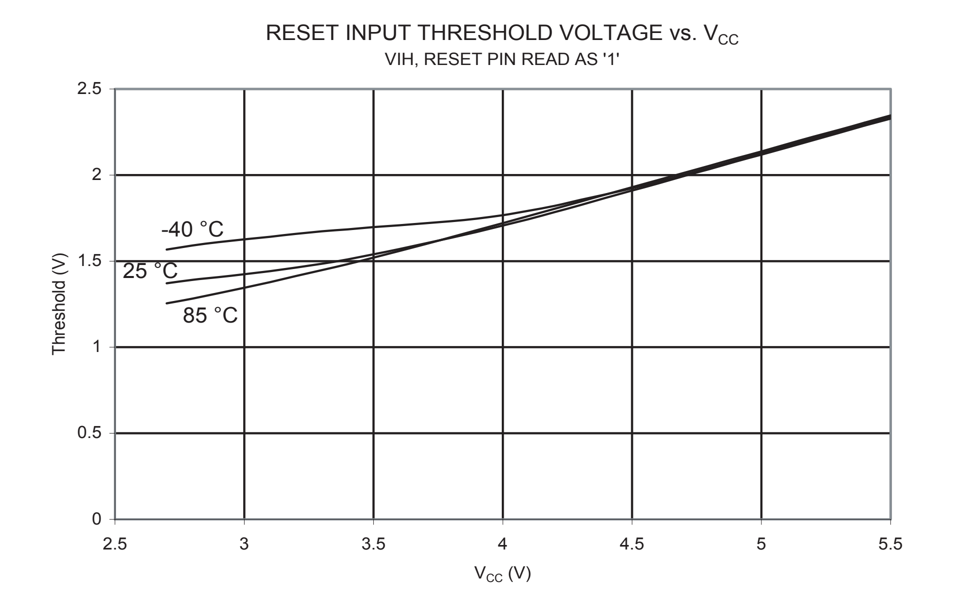


Figure 214. 复位输入阈值电压与 V_{CC} 的关系 (VIH，复位引脚读为 ‘0’)

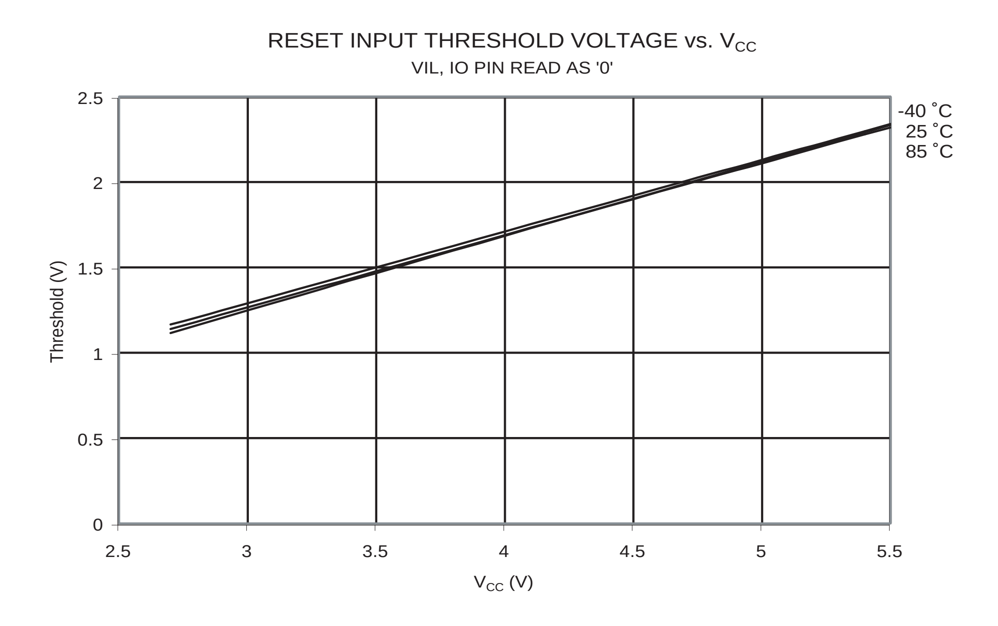


Figure 215. 复位输入引脚迟滞与 V_{CC} 的关系

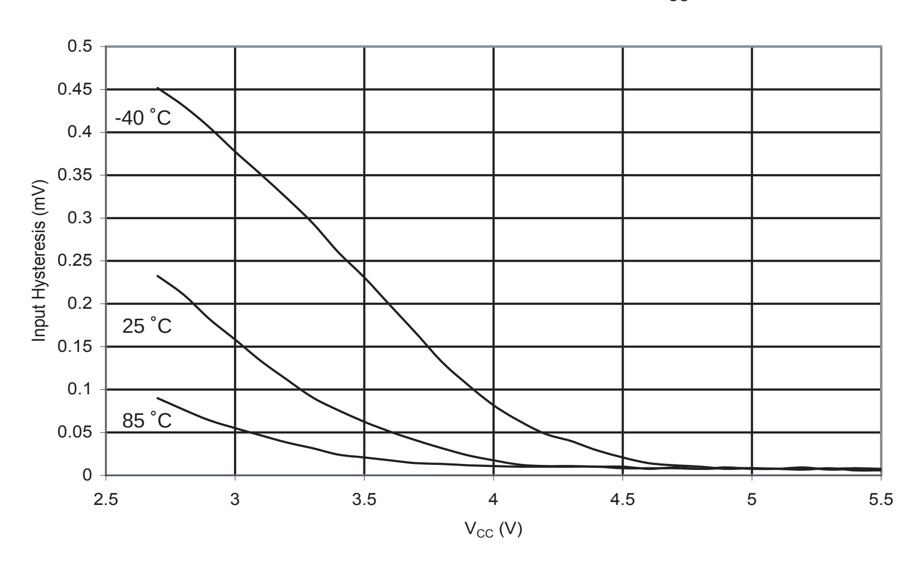
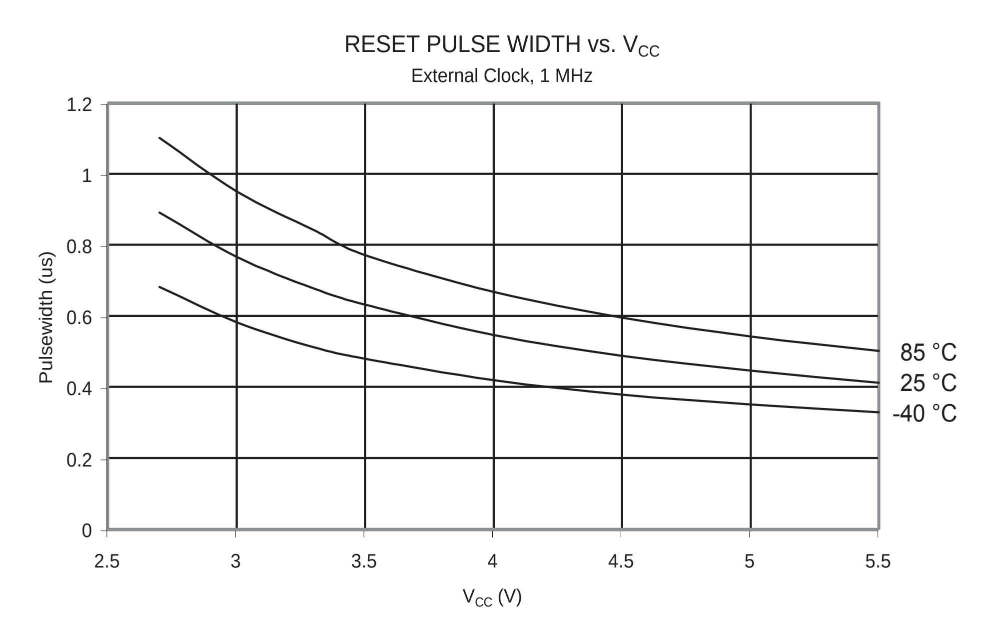


Figure 216. 复位脉宽与 V_{CC} 的关系 (外部时钟， 1 MHz)



寄存器概述

地址	名称	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	页码
($FF)	保 留	–	–	–	–	–	–	–	–	
..	保 留	–	–	–	–	–	–	–	–	
($9E)	保 留	–	–	–	–	–	–	–	–	
($9D)	UCSR1C	–	UMSEL1	UPM11	UPM10	USBS1	UCSZ11	UCSZ10	UCPOL1	174
($9C)	UDR1	USART1 I/O 数据寄存器								173
($9B)	UCSR1A	RXC1	TXC1	UDRE1	FE1	DOR1	UPE1	U2X1	MPCM1	173
($9A)	UCSR1B	RXCIE1	TXCIE1	UDRIE1	RXEN1	TXEN1	UCSZ12	RXB81	TXB81	174
($99)	UBRR1L	USART1 波特率寄存器低字节								176
($98)	UBRR1H	–	–	–	–	USART1 波特率寄存器高字节				176
($97)	保 留	–	–	–	–	–	–	–	–	
($96)	保 留	–	–	–	–	–	–	–	–	
($95)	UCSR0C	–	UMSEL0	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0	174
($94)	保 留	–	–	–	–	–	–	–	–	
($93)	保 留	–	–	–	–	–	–	–	–	
($92)	保 留	–	–	–	–	–	–	–	–	
($91)	保 留	–	–	–	–	–	–	–	–	
($90)	UBRR0H	–	–	–	–	USART0 波特率寄存器高字节				176
($8F)	保 留	–	–	–	–	–	–	–	–	
($8E)	保 留	–	–	–	–	–	–	–	–	
($8D)	保 留	–	–	–	–	–	–	–	–	
($8C)	TCCR3C	FOC3A	FOC3B	FOC3C	–	–	–	–	–	124
($8B)	TCCR3A	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	120
($8A)	TCCR3B	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	123
($89)	TCNT3H	T/C3 – 计数器寄存器高字节								125
($88)	TCNT3L	T/C3 – 计数器寄存器低字节								125
($87)	OCR3AH	T/C3 – 输出比较寄存器 A 高字节								125
($86)	OCR3AL	T/C3 – 输出比较寄存器 A 低字节								125
($85)	OCR3BH	T/C3 – 输出比较寄存器 B 高字节								125
($84)	OCR3BL	T/C3 – 输出比较寄存器 B 低字节								125
($83)	OCR3CH	T/C3– 输出比较寄存器 C 高字节								126
($82)	OCR3CL	T/C3 – 输出比较寄存器 C 低字节								125
($81)	ICR3H	T/C3 – 输入捕捉寄存器高字节								126
($80)	ICR3L	T/C3 – 输入捕捉寄存器低字节								126
($7F)	保 留	–	–	–	–	–	–	–	–	
($7E)	保 留	–	–	–	–	–	–	–	–	
($7D)	ETIMSK	–	–	TICIE3	OCIE3A	OCIE3B	TOIE3	OCIE3C	OCIE1C	127
($7C)	ETIFR	–	–	ICF3	OCF3A	OCF3B	TOV3	OCF3C	OCF1C	128
($7B)	保 留	–	–	–	–	–	–	–	–	
($7A)	TCCR1C	FOC1A	FOC1B	FOC1C	–	–	–	–	–	124
($79)	OCR1CH	T/C1 – 输出比较寄存器 C 高字节								125
($78)	OCR1CL	T/C1 – 输出比较寄存器 C 低字节								125
($77)	保 留	–	–	–	–	–	–	–	–	
($76)	保 留	–	–	–	–	–	–	–	–	
($75)	保 留	–	–	–	–	–	–	–	–	
($74)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	188
($73)	TWDR	两线串行接口数据寄存器								189
($72)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	190
($71)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	189
($70)	TWBR	两线串行接口比特率寄存器								188
($6F)	OSCCAL	振荡器标定寄存器								38
($6E)	保 留	–	–	–	–	–	–	–	–	
($6D)	XMCRA	–	SRL2	SRL1	SRL0	SRW01	SRW00	SRW11		28
($6C)	XMCRB	XMBK	–	–	–	–	XMM2	XMM1	XMM0	29
($6B)	保 留	–	–	–	–	–	–	–	–	
($6A)	EICRA	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	84
($69)	保 留	–	–	–	–	–	–	–	–	
($68)	SPMCSR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	258
($67)	保 留	–	–	–	–	–	–	–	–	
($66)	保 留	–	–	–	–	–	–	–	–	
($65)	PORTG	–	–	–	PORTG4	PORTG3	PORTG2	PORTG1	PORTG0	83
($64)	DDRG	–	–	–	DDG4	DDG3	DDG2	DDG1	DDG0	83
($63)	PING	–	–	–	PING4	PING3	PING2	PING1	PING0	83
($62)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	PORTF3	PORTF2	PORTF1	PORTF0	82

寄存器概述

地址	名称	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	页码
($61)	DDRF	DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0	83
($60)	Reserved	–	–	–	–	–	–	–	–	
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	8
$3E ($5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	11
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	11
$3C ($5C)	XDIV	XDIVEN	XDIV6	XDIV5	XDIV4	XDIV3	XDIV2	XDIV1	XDIV0	40
$3B ($5B)	RAMPZ	–	–	–	–	–	–	–	RAMPZ0	11
$3A ($5A)	EICRB	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	84
$39 ($59)	EIMSK	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	85
$38 ($58)	EIFR	INTF7	INTF6	INTF5	INTF4	INTF3	INTF	INTF1	INTF0	85
$37 ($57)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	98, 126, 144
$36 ($56)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	98, 128, 144
$35 ($55)	MCUCR	SRE	SRW10	SE	SM1	SM0	SM2	IVSEL	IVCE	28, 41, 58
$34 ($54)	MCUCSR	JTD	–	–	JTRF	WDRF	BORF	EXTRF	PORF	50, 237
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	94
$32 ($52)	TCNT0	T/C0 (8 比特)								96
$31 ($51)	OCR0	T/C0 输出比较寄存器								96
$30 ($50)	ASSR	–	–	–	–	AS0	TCN0UB	OCR0UB	TCR0UB	97
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	120
$2E ($4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	123
$2D ($4D)	TCNT1H	T/C1 – 计数器寄存器高字节								125
$2C ($4C)	TCNT1L	T/C1 – 计数器寄存器低字节								125
$2B ($4B)	OCR1AH	T/C1 – 输出比较寄存器 A 高字节								125
$2A ($4A)	OCR1AL	T/C1 – 输出比较寄存器 A 低字节								125
$29 ($49)	OCR1BH	T/C1 – 输出比较寄存器 B 高字节								125
$28 ($48)	OCR1BL	T/C1 – 输出比较寄存器 B 低字节								125
$27 ($47)	ICR1H	T/C1 – 输入捕捉寄存器高字节								126
$26 ($46)	ICR1L	T/C1 – 输入捕捉寄存器低字节								126
$25 ($45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	142
$24 ($44)	TCNT2	T/C2 (8 比特)								144
$23 ($43)	OCR2	T/C2 输出比较寄存器								144
$22 ($42)	OCDR	IDRD/OCDR7	OCDR6	OCDR5	OCDR4	OCDR3	OCDR2	OCDR1	OCDR0	234
$21 ($41)	WDTCR	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0	52
$20 ($40)	SFIOR	TSM	–	–	–	ACME	PUD	PSR0	PSR321	68, 99, 131, 210
$1F ($3F)	EEARH	–	–	–	–	EEPROM 地址寄存器高字节				18
$1E ($3E)	EEARL	EEPROM 地址寄存器低字节								18
$1D ($3D)	EEDR	EEPROM 数据寄存器								18
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	19
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	81
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	81
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	81
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	81
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	81
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	81
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	81
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	81
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	82
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	82
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	82
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	82
$0F ($2F)	SPDR	SPI 数据寄存器								153
$0E ($2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	153
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	151
$0C ($2C)	UDR0	USART0 I/O 数据寄存器								173
$0B ($2B)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0	173
$0A ($2A)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80	174
$09 ($29)	UBRR0L	USART0 波特率寄存器低字节								176
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	210
$07 ($27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	225
$06 ($26)	ADCSRA	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	227
$05 ($25)	ADCH	ADC 数据寄存器高字节								228
$04 ($24)	ADCL	ADC 数据寄存器低字节								228
$03 ($23)	PORTE	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0	82
$02 ($22)	DDRE	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0	82



寄存器概述

地址	名称	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	页码
$01 ($21)	PINE	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0	82
$00 ($20)	PINF	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0	83

Notes: 1. 为了保持与后续器件的兼容性，在访问寄存器时保留的位应写 0。保留的 I/O 存储器地址则不应该访问。
2. 某些状态标志位清零是通过写入逻辑1来实现的。CBI和SBI指令操作于I/O寄存器的所有位，并对置位的位进行写回1的操作，从而将该标志清零。 CBI 和 SBI 指令只可以操作 $00 到 $1F 的寄存器。

指令集概述

助记符	操作数	说明	操作	影响的标志位	助记符
算术和逻辑指令					**算术和逻辑指令**
ADD	Rd, Rr	两个寄存器相加	Rd ← Rd + Rr	Z,C,N,V,H	ADD
ADC	Rd, Rr	两个寄存器带进位位相加	Rd ← Rd + Rr + C	Z,C,N,V,H	ADC
ADIW	Rdl,K	立即数与字相加	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	ADIW
SUB	Rd, Rr	两个寄存器相减	Rd ← Rd - Rr	Z,C,N,V,H	SUB
SUBI	Rd, K	寄存器与常数相减	Rd ← Rd - K	Z,C,N,V,H	SUBI
SBC	Rd, Rr	两个寄存器带进位位相减	Rd ← Rd - Rr - C	Z,C,N,V,H	SBC
SBCI	Rd, K	寄存器与常数及进位位相减	Rd ← Rd - K - C	Z,C,N,V,H	SBCI
SBIW	Rdl,K	字与立即数相减	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	SBIW
AND	Rd, Rr	两个寄存器逻辑与	Rd ← Rd • Rr	Z,N,V	AND
ANDI	Rd, K	寄存器与常数逻辑与	Rd ← Rd • K	Z,N,V	ANDI
OR	Rd, Rr	两个寄存器逻辑或	Rd ← Rd v Rr	Z,N,V	OR
ORI	Rd, K	寄存器与常数逻辑或	Rd ← Rd v K	Z,N,V	ORI
EOR	Rd, Rr	两个寄存器异或	Rd ← Rd ⊕ Rr	Z,N,V	EOR
COM	Rd	1 的补码	Rd ← $FF − Rd	Z,C,N,V	COM
NEG	Rd	2 的补码	Rd ← $00 − Rd	Z,C,N,V,H	NEG
SBR	Rd,K	寄存器的某些位置位	Rd ← Rd v K	Z,N,V	SBR
CBR	Rd,K	寄存器的某些位清零	Rd ← Rd • ($FF - K)	Z,N,V	CBR
INC	Rd	加一	Rd ← Rd + 1	Z,N,V	INC
DEC	Rd	减一	Rd ← Rd − 1	Z,N,V	DEC
TST	Rd	测试为 0 或负	Rd ← Rd • Rd	Z,N,V	TST
CLR	Rd	清零寄存器	Rd ← Rd ⊕ Rd	Z,N,V	CLR
SER	Rd	置位寄存器	Rd ← $FF	None	SER
MUL	Rd, Rr	无符号数相乘	R1:R0 ← Rd x Rr	Z,C	MUL
MULS	Rd, Rr	有符号数相乘	R1:R0 ← Rd x Rr	Z,C	MULS
MULSU	Rd, Rr	有符号数与无符号数相乘	R1:R0 ← Rd x Rr	Z,C	MULSU
FMUL	Rd, Rr	无符号小数相乘	R1:R0 ← (Rd x Rr) << 1	Z,C	FMUL
FMULS	Rd, Rr	有符号小数相乘	R1:R0 ← (Rd x Rr) << 1	Z,C	FMULS
FMULSU	Rd, Rr	无符号小数与有符号小数相乘	R1:R0 ← (Rd x Rr) << 1	Z,C	FMULSU
跳转指令					**跳转指令**
RJMP	k	相对跳转	PC ← PC + k + 1	无	RJMP
IJMP		间接跳转到 (Z)	PC ← Z	无	IJMP
JMP	k	直接跳转	PC ← k	无	JMP
RCALL	k	相对子程序调用	PC ← PC + k + 1	无	RCALL
ICALL		相对调用到 (Z)	PC ← Z	无	ICALL
CALL	k	直接子程序调用	PC ← k	无	CALL
RET		子程序返回	PC ← STACK	无	RET
RETI		中断返回	PC ← STACK	I	RETI
CPSE	Rd,Rr	比较，相等则跳过下一条指令	if (Rd = Rr) PC ← PC + 2 or 3	无	CPSE
CP	Rd,Rr	比较	Rd − Rr	Z, N,V,C,H	CP
CPC	Rd,Rr	带进位位比较	Rd − Rr − C	Z, N,V,C,H	CPC
CPI	Rd,K	比较寄存器与立即数	Rd − K	Z, N,V,C,H	CPI
SBRC	Rr, b	若寄存器的某一位清零即跳转	if (Rr(b)=0) PC ← PC + 2 or 3	无	SBRC
SBRS	Rr, b	若寄存器的某一位置位即跳转	if (Rr(b)=1) PC ← PC + 2 or 3	无	SBRS
SBIC	P, b	若 I/O 寄存器的某一位清零即跳转	if (P(b)=0) PC ← PC + 2 or 3	无	SBIC
SBIS	P, b	若 I/O 寄存器的某一位置位即跳转	if (P(b)=1) PC ← PC + 2 or 3	无	SBIS
BRBS	s, k	若状态标志置位即跳转	if (SREG(s) = 1) then PC←PC+k + 1	无	BRBS
BRBC	s, k	若状态标志清零即跳转	if (SREG(s) = 0) then PC←PC+k + 1	无	BRBC
BREQ	k	相等即跳转	if (Z = 1) then PC ← PC + k + 1	无	BREQ
BRNE	k	不相等即跳转	if (Z = 0) then PC ← PC + k + 1	无	BRNE
BRCS	k	进位标志置位即跳转	if (C = 1) then PC ← PC + k + 1	无	BRCS
BRCC	k	进位标志清零即跳转	if (C = 0) then PC ← PC + k + 1	无	BRCC
BRSH	k	相同或更大即跳转	if (C = 0) then PC ← PC + k + 1	无	BRSH
BRLO	k	小于即跳转	if (C = 1) then PC ← PC + k + 1	无	BRLO
BRMI	k	负数即跳转	if (N = 1) then PC ← PC + k + 1	无	BRMI
BRPL	k	正数即跳转	if (N = 0) then PC ← PC + k + 1	无	BRPL
BRGE	k	有符号数大于 / 等于即跳转	if (N ⊕ V= 0) then PC ← PC + k + 1	无	BRGE
BRLT	k	有符号数小于 0 即跳转	if (N ⊕ V= 1) then PC ← PC + k + 1	无	BRLT
BRHS	k	半进位标志置位即跳转	if (H = 1) then PC ← PC + k + 1	无	BRHS
BRHC	k	半进位标志清零即跳转	if (H = 0) then PC ← PC + k + 1	无	BRHC
BRTS	k	T 标志置位即跳转	if (T = 1) then PC ← PC + k + 1	无	BRTS
BRTC	k	T 标志清零即跳转	if (T = 0) then PC ← PC + k + 1	无	BRTC
BRVS	k	溢出标志置位即跳转	if (V = 1) then PC ← PC + k + 1	无	BRVS
BRVC	k	溢出标志清零即跳转	if (V = 0) then PC ← PC + k + 1	无	BRVC



指令集概述

助记符	操作数	说明	操作	影响的标志位	助记符
BRIE	k	中断使能即跳转	if (I = 1) then PC ← PC + k + 1	无	BRIE
BRID	k	中断禁止即跳转	if (I = 0) then PC ← PC + k + 1	无	BRID
数据传输指令					**数据传输指令**
MOV	Rd, Rr	寄存器之间数据转移	Rd ← Rr	无	MOV
MOVW	Rd, Rr	拷贝寄存器字	Rd+1:Rd ← Rr+1:Rr	无	MOVW
LDI	Rd, K	加载立即数	Rd ← K	无	LDI
LD	Rd, X	间接加载	Rd ← (X)	无	LD
LD	Rd, X+	间接加载并执行后加操作	Rd ← (X), X ← X + 1	无	LD
LD	Rd, - X	间接加载并执行预减操作	X ← X - 1, Rd ← (X)	无	LD
LD	Rd, Y	间接加载	Rd ← (Y)	无	LD
LD	Rd, Y+	间接加载并执行后加操作 .	Rd ← (Y), Y ← Y + 1	无	LD
LD	Rd, - Y	间接加载并执行预减操作	Y ← Y - 1, Rd ← (Y)	无	LD
LDD	Rd,Y+q	带偏移量的间接加载	Rd ← (Y + q)	无	LDD
LD	Rd, Z	间接加载	Rd ← (Z)	无	LD
LD	Rd, Z+	间接加载并执行后加操作	Rd ← (Z), Z ← Z+1	无	LD
LD	Rd, -Z	间接加载并执行预减操作	Z ← Z - 1, Rd ← (Z)	无	LD
LDD	Rd, Z+q	带偏移量的间接加载	Rd ← (Z + q)	无	LDD
LDS	Rd, k	从 SRAM 中直接加载	Rd ← (k)	无	LDS
ST	X, Rr	间接存储	(X) ← Rr	无	ST
ST	X+, Rr	间接存储并执行后加操作	(X) ← Rr, X ← X + 1	无	ST
ST	- X, Rr	间接存储并执行预减操作	X ← X - 1, (X) ← Rr	无	ST
ST	Y, Rr	间接存储	(Y) ← Rr	无	ST
ST	Y+, Rr	间接存储并执行后加操作	(Y) ← Rr, Y ← Y + 1	无	ST
ST	- Y, Rr	间接存储并执行预减操作	Y ← Y - 1, (Y) ← Rr	无	ST
STD	Y+q,Rr	带偏移量的间接存储	(Y + q) ← Rr	无	STD
ST	Z, Rr	间接存储	(Z) ← Rr	无	ST
ST	Z+, Rr	间接存储并执行后加操作	(Z) ← Rr, Z ← Z + 1	无	ST
ST	-Z, Rr	间接存储并执行预减操作	Z ← Z - 1, (Z) ← Rr	无	ST
STD	Z+q,Rr	带偏移量的间接存储	(Z + q) ← Rr	无	STD
STS	k, Rr	直接存储到 SRAM	(k) ← Rr	无	STS
LPM		加载程序存储器	R0 ← (Z)	无	LPM
LPM	Rd, Z	加载程序存储器	Rd ← (Z)	无	LPM
LPM	Rd, Z+	加载程序存储器并执行后加操作	Rd ← (Z), Z ← Z+1	无	LPM
ELPM		扩展的加载程序存储器操作	R0 ← (RAMPZ:Z)	无	ELPM
ELPM	Rd, Z	扩展的加载程序存储器操作	Rd ← (RAMPZ:Z)	无	ELPM
ELPM	Rd, Z+	扩展的加载程序存储器操作并执行后加操作	Rd ← (RAMPZ:Z), RAMPZ:Z ← RAMPZ:Z+1	无	ELPM
SPM		存储程序存储器	(Z) ← R1:R0	无	SPM
IN	Rd, P	读入端口数据	Rd ← P	无	IN
OUT	P, Rr	将数据输出到端口	P ← Rr	无	OUT
PUSH	Rr	将寄存器的数值推入堆栈	STACK ← Rr	无	PUSH
POP	Rd	将寄存器的数值弹出堆栈	Rd ← STACK	无	POP
位和位测试指令					**位和位测试指令**
SBI	P,b	置位 I/O 寄存器的某一位	I/O(P,b) ← 1	无	SBI
CBI	P,b	清除 I/O 寄存器的某一位	I/O(P,b) ← 0	无	CBI
LSL	Rd	逻辑左移	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	LSL
LSR	Rd	逻辑右移	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	LSR
ROL	Rd	通过进位位向左循环	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	ROL
ROR	Rd	通过进位位向右循环	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	ROR
ASR	Rd	算术右移	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	ASR
SWAP	Rd	高 4 位与低 4 位交换	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	无	SWAP
BSET	s	置位标志位	SREG(s) ← 1	SREG(s)	BSET
BCLR	s	清零标志位	SREG(s) ← 0	SREG(s)	BCLR
BST	Rr, b	将寄存器的某一位保存到 T	T ← Rr(b)	T	BST
BLD	Rd, b	将 T 加载到寄存器的某一位	Rd(b) ← T	无	BLD
SEC		置位进位位	C ← 1	C	SEC
CLC		清零进位位	C ← 0	C	CLC
SEN		置位负数标志位	N ← 1	N	SEN
CLN		清零负数标志位	N ← 0	N	CLN
SEZ		置位 0 标志位	Z ← 1	Z	SEZ
CLZ		清零 0 标志位	Z ← 0	Z	CLZ
SEI		全局中断标志使能	I ← 1	I	SEI
CLI		全局中断标志禁止	I ← 0	I	CLI
SES		置位符号测试标志位	S ← 1	S	SES
CLS		清零符号测试标志位	S ← 0	S	CLS

指令集概述

助记符	操作数	说明	操作	影响的标志位	助记符
SEV		置位 2 的补码溢出标志	V ← 1	V	SEV
CLV		清除 2 的补码溢出标志	V ← 0	V	CLV
SET		置位 SREG 的 T 标志	T ← 1	T	SET
CLT		清除 SREG 的 T 标志	T ← 0	T	CLT
SEH		置位 SREG 的半进位标志	H ← 1	H	SEH
CLH		清除 SREG 的半进位标志	H ← 0	H	CLH
MCU 控制指令					**MCU 控制指令**
NOP		无操作		无	NOP
SLEEP		睡眠	(参见有关的睡眠说明)	无	SLEEP
WDR		看门狗复位	(参见有关的看门狗定时器说明)	无	WDR
BREAK		Break	只用于片上调试	无	BREAK



定货信息

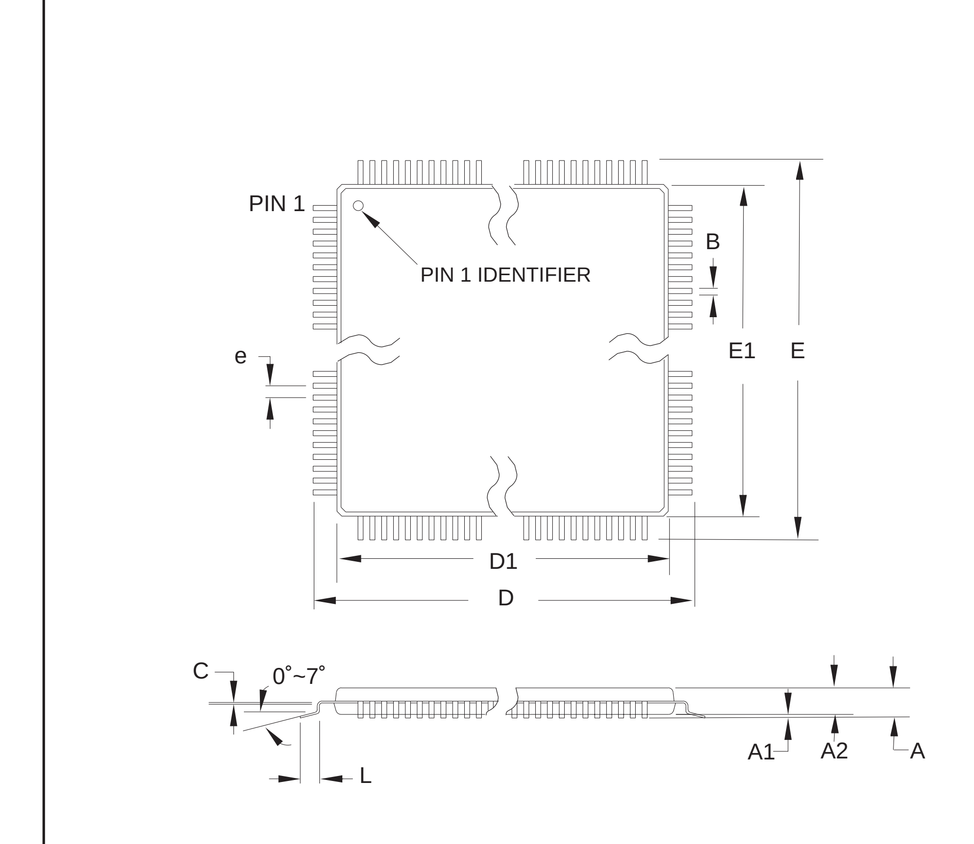
工作速度 (MHz)	工作电压	定货号	封装	工作范围
8	2.7 - 5.5V	ATmega128L-8AC ATmega128L-8MC	64A 64M1	商业级 (0°C - 70°C)
		ATmega128L-8AI ATmega128L-8AJ(2) ATmega128L-8MI ATmega128L-8MJ(2)	64A 64A 64M1 64M1	工业级 (-40°C - 85°C)
16	4.5 - 5.5V	ATmega128-16AC ATmega128-16MC	64A 64M1	商业级 (0°C - 70°C)
		ATmega128-16AI ATmega128-16AJ(2) ATmega128-16MI ATmega128-16MJ(2)	64A 64A 64M1 64M1	工业级 (-40°C - 85°C)

Notes: 1. 产品也可以 wafer 的形式提供，订货信息细节以及最小定货量请与 Atmel 当地机构联系。
2. 可选无铅封装。

封装类型	
64A	64- 引线，薄 (1.0 mm) TQFP 封装
64M1	64- 焊垫， 9 x 9 x 1.0 mm 大小，线距 0.50 mm， MLF 封装

封装信息

64A



COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	1.20	
A1	0.05	–	0.15	
A2	0.95	1.00	1.05	
D	15.75	16.00	16.25	
D1	13.90	14.00	14.10	Note 2
E	15.75	16.00	16.25	
E1	13.90	14.00	14.10	Note 2
B	0.30	–	0.45	
C	0.09	–	0.20	
L	0.45	–	0.75	
e	0.80 TYP			

Notes:
1. This package conforms to JEDEC reference MS-026, Variation AEB.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25 mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.10 mm maximum.

10/5/2001

ATMEL 2325 Orchard Parkway San Jose, CA 95131	TITLE **64A,** 64-lead, 14 x 14 mm Body Size, 1.0 mm Body Thickness, 0.8 mm Lead Pitch, Thin Profile Plastic Quad Flat Package (TQFP)	DRAWING NO. 64A	REV. B



64M1

TOP VIEW

Marked Pin# 1 ID

D

E

SIDE VIEW

C SEATING PLANE

A1

A

0.08 C

BOTTOM VIEW

Pin #1 Corner

L

D2

E2

b

e

1
2
3

COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	0.80	0.90	1.00	
A1	–	0.02	0.05	
b	0.23	0.25	0.28	
D		9.00 BSC		
D2	5.20	5.40	5.60	
E		9.00 BSC		
E2	5.20	5.40	5.60	
e		0.50 BSC		
L	0.35	0.40	0.45	

Notes: 1. JEDEC Standard MO-220, Fig. 1, VMMD.

01/15/03

	TITLE	DRAWING NO.	REV.
ATMEL 2325 Orchard Parkway San Jose, CA 95131	**64M1**, 64-pad, 9 x 9 x 1.0 mm Body, Lead Pitch 0.50 mm Micro Lead Frame Package (MLF)	64M1	C

勘误表

本节的版本号与 ATmega128 器件的版本号相同。

ATmega128 Rev. I

- **改变 XDIV 寄存器时需稳定时间**
- **改变 OSCCAL 寄存器时需稳定时间**

1. 改变 XDIV 寄存器时需稳定时间

通过设置 XDIV 寄存器将时钟源频率提高超过 2% 时，器件可能执行一些错误指令。

解决方案

即使频率改变，NOP 指令也会正确执行。因此，在频率改变后执行 8 条 NOP 指令。通过下面程序进行：

1.SREG 寄存器 I 位清零。

2. 在 XDIV 寄存器中设置新的预分频因子。

3.执行 8 条 NOP 指令。

4.SREG 寄存器 I 位置位。

这样就可以保证所有指令正确执行。

汇编代码例程：

```
CLI                 ; 全局中断使能清零
OUT  XDIV, temp     ; 设置新的预分频因子
NOP                 ; 无操作
NOP                 ; 无操作
NOP                 ; 无操作
NOP                 ; 无操作
NOP                 ; 无操作
NOP                 ; 无操作
NOP                 ; 无操作
NOP                 ; 无操作
SEI                 ; 全局中断使能置位
```

2. 改变 OSCCAL 寄存器时需稳定时间

通过设置 OSCCAL 寄存器将时钟源频率提高超过 2% 时，器件可能执行一些错误指令。

解决方案

方法如勘误表 1。

下面是针对 JTAG 指令 IDCODE 问题的解决方案。

IDCODE 屏蔽了从 TDI 输入的数据

共用可选 JTAG 指令 IDCODE 在 IEEE1149.1 中不能正确执行； ”1” 取代 TDI 输入被扫描进入移位寄存器，且移出器件 ID 寄存器。因此，从边界扫描链上前面器件捕获的数据全部丢失，被 1 所代替，在 Update-DR 时全被 1 所代替。

若 ATmega128 是扫描链上唯一的器件，则不会发现这个问题。

解决方法

通过使用 IDCODE 命令或通过进入 TAP 控制器的 Test-Logic-Reset 状态来选择器件 ID 寄存器，然后读取其内容，以及后续器件的数据。注意，扫描时数据不能进入后续器件，但可以进入前面的器件。读取扫描链中 ATmega128 前面器件的器件 ID 寄存器时对 ATmega128 发送 BYPASS 命令。不从边界扫描器的后续器件中读取数据，当器件 ID 寄存器为 ATmega128 时不对后续器件上传数据。注意 IDCODE 指令不是 TAP 控制器 Test-Logic-Reset 状态的默认指令。



其他解决方法

如果必须同时获取扫描链中所有器件的 ID ,那么 ATmega128 应该是扫描链中的第一个器件。只要 IDCODE 指令在 JTAG 指令寄存器中,Update-DR 对后续器件就不工作,但器件 ID 寄存器无法上传。

ATmega128 Rev. H

- **改变 XDIV 寄存器时需稳定时间**
- **改变 OSCCAL 寄存器时需稳定时间**

1. 改变 XDIV 寄存器时需稳定时间

通过设置 XDIV 寄存器将时钟源频率提高超过 2% 时,器件可能执行一些错误指令。

解决方案

即使频率改变,NOP 指令也会正确执行。因此,在频率改变后执行 8 条 NOP 指令。通过下面程序进行:

1.SREG 寄存器 I 位清零。

2. 在 XDIV 寄存器中设置新的预分频因子。

3.执行 8 条 NOP 指令。

4.SREG 寄存器 I 位置位。

这样就可以保证所有指令正确执行。

汇编代码例程:

```
CLI                 ; 全局中断使能清零
OUT  XDIV, temp     ; 设置新的预分频因子
NOP                 ; 无操作
NOP                 ; 无操作
NOP                 ; 无操作
NOP                 ; 无操作
NOP                 ; 无操作
NOP                 ; 无操作
NOP                 ; 无操作
NOP                 ; 无操作
SEI                 ; 全局中断使能置位
```

2. 改变 OSCCAL 寄存器时需稳定时间

通过设置 OSCCAL 寄存器将时钟源频率提高超过 2% 时,器件可能执行一些错误指令。

解决方案

方法如勘误表 1。

下面是针对 JTAG 指令 IDCODE 问题的解决方案。

IDCODE 屏蔽了从 TDI 输入的数据

共用可选 JTAG 指令 IDCODE 在 IEEE1149.1 中不能正确执行; ”1” 取代 TDI 输入被扫描进入移位寄存器,且移出器件 ID 寄存器。因此,从边界扫描链上前面器件捕获的数据全部丢失,被 1 所代替,在 Update-DR 时全被 1 所代替。

若 ATmega128 是扫描链上唯一的器件,则不会发现这个问题。

解决方法

通过使用 IDCODE 命令或通过进入 TAP 控制器的 Test-Logic-Reset 状态来选择器件 ID 寄存器，然后读取其内容，以及后续器件的数据。注意，扫描时数据不能进入后续器件，但可以进入前面的器件。读取扫描链中 ATmega128 前面器件的器件 ID 寄存器时对 ATmega128 发送 BYPASS 命令。不从边界扫描器的后续器件中读取数据，当器件 ID 寄存器为 ATmega128 时不对后续器件上传数据。注意 IDCODE 指令不是 TAP 控制器 Test-Logic-Reset 状态的默认指令。

其他解决方法

如果必须同时获取扫描链中所有器件的 ID，那么 ATmega128 应该是扫描链中的第一个器件。只要 IDCODE 指令在 JTAG 指令寄存器中，Update-DR 对后续器件就不工作，但器件 ID 寄存器无法上传。

ATmega128 Rev. G

- **改变 XDIV 寄存器时需稳定时间**
- **改变 OSCCAL 寄存器时需稳定时间**

1. 改变 XDIV 寄存器时需稳定时间

通过设置 XDIV 寄存器将时钟源频率提高超过 2% 时，器件可能执行一些错误指令。

解决方案

即使频率改变，NOP 指令也会正确执行。因此，在频率改变后执行 8 条 NOP 指令。通过下面程序进行：

1.SREG 寄存器 I 位清零。

2. 在 XDIV 寄存器中设置新的预分频因子。

3.执行 8 条 NOP 指令。

4.SREG 寄存器 I 位置位。

这样就可以保证所有指令正确执行。

汇编代码例程：

```
CLI                ; 全局中断使能清零
OUT  XDIV, temp    ; 设置新的预分频因子
NOP                ; 无操作
NOP                ; 无操作
NOP                ; 无操作
NOP                ; 无操作
NOP                ; 无操作
NOP                ; 无操作
NOP                ; 无操作
NOP                ; 无操作
SEI                ; 全局中断使能置位
```

2. 改变 OSCCAL 寄存器时需稳定时间

通过设置 OSCCAL 寄存器将时钟源频率提高超过 2% 时，器件可能执行一些错误指令。

解决方案

方法如勘误表 1。

下面是针对 JTAG 指令 IDCODE 问题的解决方案。

IDCODE 屏蔽了从 TDI 输入的数据



共用可选 JTAG 指令 IDCODE 在 IEEE1149.1 中不能正确执行；”1” 取代 TDI 输入被扫描进入移位寄存器，且移出器件 ID 寄存器。因此，从边界扫描链上前面器件捕获的数据全部丢失，被 1 所代替，在 Update-DR 时全被 1 所代替。

若 ATmega128 是扫描链上唯一的器件，则不会发现这个问题。

解决方法

通过使用 IDCODE 命令或通过进入 TAP 控制器的 Test-Logic-Reset 状态来选择器件 ID 寄存器，然后读取其内容，以及后续器件的数据。注意，扫描时数据不能进入后续器件，但可以进入前面的器件。读取扫描链中 ATmega128 前面器件的器件 ID 寄存器时对 ATmega128 发送 BYPASS 命令。不从边界扫描器的后续器件中读取数据，当器件 ID 寄存器为 ATmega128 时不对后续器件上传数据。注意 IDCODE 指令不是 TAP 控制器 Test-Logic-Reset 状态的默认指令。

其他解决方法

如果必须同时获取扫描链中所有器件的 ID，那么 ATmega128 应该是扫描链中的第一个器件。只要 IDCODE 指令在 JTAG 指令寄存器中，Update-DR 对后续器件就不工作，但器件 ID 寄存器无法上传。

ATmega128 Rev. F

- **改变 XDIV 寄存器时需稳定时间**
- **改变 OSCCAL 寄存器时需稳定时间**

1. 改变 XDIV 寄存器时需稳定时间

通过设置 XDIV 寄存器将时钟源频率提高超过 2% 时，器件可能执行一些错误指令。

解决方案

即使频率改变， NOP 指令也会正确执行。因此，在频率改变后执行 8 条 NOP 指令。通过下面程序进行：

1.SREG 寄存器 I 位清零。

2. 在 XDIV 寄存器中设置新的预分频因子。

3.执行 8 条 NOP 指令。

4.SREG 寄存器 I 位置位。

这样就可以保证所有指令正确执行。

汇编代码例程：

```
CLI                ; 全局中断使能清零
OUT  XDIV, temp    ; 设置新的预分频因子
NOP                ; 无操作
NOP                ; 无操作
NOP                ; 无操作
NOP                ; 无操作
NOP                ; 无操作
NOP                ; 无操作
NOP                ; 无操作
NOP                ; 无操作
SEI                ; 全局中断使能置位
```

2. 改变 OSCCAL 寄存器时需稳定时间

通过设置 OSCCAL 寄存器将时钟源频率提高超过 2% 时，器件可能执行一些错误指令。

解决方案

方法如勘误表 1。

下面是针对 JTAG 指令 IDCODE 问题的解决方案。

IDCODE 屏蔽了从 TDI 输入的数据

共用可选 JTAG 指令 IDCODE 在 IEEE1149.1 中不能正确执行； ”1” 取代 TDI 输入被扫描进入移位寄存器，且移出器件 ID 寄存器。因此，从边界扫描链上前面器件捕获的数据全部丢失，被 1 所代替，在 Update-DR 时全被 1 所代替。

若 ATmega128 是扫描链上唯一的器件，则不会发现这个问题。

解决方法

通过使用 IDCODE 命令或通过进入 TAP 控制器的 Test-Logic-Reset 状态来选择器件 ID 寄存器，然后读取其内容，以及后续器件的数据。注意，扫描时数据不能进入后续器件，但可以进入前面的器件。读取扫描链中 ATmega128 前面器件的器件 ID 寄存器时对 ATmega128 发送 BYPASS 命令。不从边界扫描器的后续器件中读取数据，当器件 ID 寄存器为 ATmega128 时不对后续器件上传数据。注意 IDCODE 指令不是 TAP 控制器 Test-Logic-Reset 状态的默认指令。

其他解决方法

如果必须同时获取扫描链中所有器件的 ID，那么 ATmega128 应该是扫描链中的第一个器件。只要 IDCODE 指令在 JTAG 指令寄存器中，Update-DR 对后续器件就不工作，但器件 ID 寄存器无法上传。



ATmega128 数据手册修改日志

本节所指的页号为本文的相关页码；修订号则为文档的修订号。

从版本 Rev. 2467K-03/04 到版本 Rev.2467L-05/04 的修改

1. 删除 “ 初稿 ” 与 “TBD” ，用 ICPx 替代 ICx。
2. 更新 P 35Table 8 , P 47Table 19 , P 53Table 22 , P 226Table 96 , P 280Table 126 , P 284Table 128 , P 301Table 132 及 P 303Table 134 。
3. 更新 P 23“ 外部存储器接口 ” 。
4. 更新 P 236“ 器件识别寄存器 ” 。
5. 更新 P 299“ 电气特性 ” 。
6. 更新 P 305“ 交流特性 ” 。
7. 更新 P 313“ATmega128 典型特性 ” 。
8. 更新 P 348“ 定货信息 ” 。

从版本 Rev. 2467J-12/03 到版本 Rev.2467K-03/04 的修改

1. 更新 P 351“ 勘误表 ” 。

从版本 Rev. 2467I-09/03 到版本 Rev.2467J-12/03 的修改

1. 更新 P 38“ 标定的片内 RC 振荡器 ” 。

从版本 Rev. 2467H-02/03 到版本 Rev. 2467I-09/03 的修改

1. 更新 P 40“XTAL 分频控制寄存器 - XDIV” 中注释。
2. 更新 P 45“JTAG 接口与片内调试系统 ” 。
3. 更新 P 47Table 19 中 V_{BOT} (BODLEVEL = 1) 值。
4. 更新 P 229“ 测试访问端口 - TAP” 中 JTAGEN 部分。
5. 更新 JTD 位说明。
6. 在 P 270Table 118 中添加关于 JTAGEN 熔丝位的注释。
7. 更新 P 299“ 直流特性 ” 中 R_{PU}。
8. 在 P 351“ 勘误表 ” 中添加 IDCODE 指令的其他解决方案。

从版本 Rev. 2467G-09/02 到版本 Rev. 2467H-02/03 的修改

1. 修改 SFIOR 寄存器中两预分频位的名称。
2. 在 P 296“对Flash进行编程” 与 P 297“对EEPROM进行编程” 中加入芯片擦除作为第一步。
3. 删除 “ 多功能振荡器 ” 使用注释及 “32 kHz 晶振 ” 使用注释。

4. 修正 P 113Figure 52 中 OCn 波形。
5. 定时器 1 细节的修正。
6. 对 定时器 0 与定时器 2 加入关于 PWM 对称的内容。
7. TWI 细节的修正。
8. Added reference to P 273Table 124 中添加SPI串行编程与自编程中Flash页尺寸的内容。
9. P 261“装载临时缓冲器(页加载)” 中添加关于在SPM页载入时写EEPROM的注释。
10. 删除 ADHSM。
11. 添加 P 22“ 掉电休眠模式下 EEPROM 的写入 ” 部分。
12. 更新 P 349“ 封装信息 ” 中图。

从版本 Rev. 2467F-09/02 到版本 Rev. 2467G-09/02 的修改

1. 改变 Flash 使用寿命为 10,000 次写 / 擦除周期

从版本 Rev. 2467E-04/02 到版本 Rev. 2467F-09/02 的修改

1. 添加 64 焊垫 MLF 封装，更新 P 348“ 定货信息 ” 。
2. 添加 P 30“ 使用外部的存储器 (小于 64KB)” 部分。
3. 添加 P 34“ 默认时钟源 ” 部分。
4. 将 SPMCSR 改为 SPMCSR。
5. 当使用外部时钟时 ,改变频率有一些限制 ,见 P 39“ 外部时钟 ” 与 P 301Table 131, “ 外部时钟驱动 ,” 。
6. 在 P 44“ 最小化功耗 ” 中加入关于 OCD 系统与功耗的小节。
7. 修改排版 (WGM 位设置):
   P 90“ 快速 PWM 模式 ” (T/C0)。
   P 92“ 相位修正 PWM 模式 ” (T/C0)。
   P 136“ 快速 PWM 模式 ” (T/C2)。
   P 138“ 相位修正 PWM 模式 ” (T/C2)。
8. 修正 P 176Table 81 (USART)。
9. 修正 P 243Table 102 (边界扫描)。
10. 更新 P 299“ 直流特性 ” 中 Vil 参数。

从版本 Rev. 2467D-03/02 到版本 Rev. 2467E-04/02 的修改

1. 更新 P 313“ATmega128 典型特性 ” 中特征数据部分。
2. 将下列表格更新 :



P 47Table 19 、 P 51Table 20 、 P 143Table 68 、 P 243Table 102 及 Table 136 on page 328。

3. **更新 OSCCAL 标定字节的说明**

   在手册中未说明选择 2、4、 8 MHz 振荡器时如何使用标定字节，在下面加入：

   P 38“ 振荡器标定寄存器 - OSCCAL” 及 P 270“ 标定字节 ” 。

从版本 Rev. 2467C-02/02 到版本 Rev. 2467D-03/02 的修改

1. **P 4“ATmega103 兼容模式 ” 中加入更多信息。**
2. **更新 P 20Table 2, “EEPROM 编程时间 ,” 。**
3. **更新 P 34Table 7，Table 9、 P 36Table 10，P 37Table 12，P 38Table 14 及 P 39Table 16 中的典型启动时间。**
4. **更新 P 53Table 22 中典型 WDT 时间溢出。**
5. **修正 P 227“ADC 控制和状态寄存器 A - ADCSRA” 中 ADSC 位说明。**
6. **P 224“ADC 转换结果 ” 进一步说明对 ADC 差分结果如何进行极性检测。**
7. **修正 JTAG 版本号。**
8. **P 260“在自编程时访问Flash”，P 261“通过SPM执行页擦除” 及P 261“执行页写操作 ” 中进一步说明 SPM(使用 RAMPZ) 时的寻址。**
9. **在 P 270Table 118 中加入关于 OCDEN 熔丝位的注释。**
10. **更新编程图：**

    更新 P 272Figure 135 与 P 282Figure 144 说明编程模式时 AVCC 必须连接。 P 278Figure 139 加入对熔丝位编程的说明。
11. **加入关于 PROG_PAGELOAD 与 PROG_PAGEREAD 指令使用的注释。**
12. **在 P 313“ATmega128 典型特性 ” 中加入标定 RC 振荡器特性曲线。**
13. **更新 “ 两线串行接口 TWI” 部分。**

    加入关于掉电模式时 TWI 操作及低 TWBRR 值下 TWI 作为主机的说明。P 186“ 比特率发生器单元 ” 最后加入注释。P 187“ 地址匹配单元 ” 最后加入说明。
14. **在 P 40“XTAL 分频控制寄存器 - XDIV” 中加入关于 T/C0 与时钟结合使用的注释。**

从版本 Rev. 2467B-09/01 到版本 Rev. 2467C-02/02 的修改

1. **修正关于端口 G 第二功能的说明**

   修正 P 79“ 端口 G 的第二功能 ” 中 TOSC1 与 TOSC2 的说明。
2. **在 rev. F 与 rev. G 中加入 JTAG 版本号**

   更新 Table 100。

3 **加入一些预测极限及特性数据**

   在下列表与页中删除 TBD：

   P 47Table 19 、P 51Table 20 、P 299“ 直流特性 ” 、P 301Table 131 、P 303Table 134 及 Table 136。

4. **修正 P 348“ 定货信息 ” 。**

5. **在 P 313“ATmega128 典型特性 ” 中加入特性数据。**

6. **删除退出 JTAG 编程模式的其他算法。**
   P 295 “ 退出编程模式 ”

7. **加入在 JTAG 编程模式下如何访问扩展熔丝字节**
   见 P 298“ 对熔丝位进行编程 ” 及 P 298“ 读取熔丝位和锁定位 ” 。





Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/ High Speed Converters/RF Data-com
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature

2467L–AVR–05/04